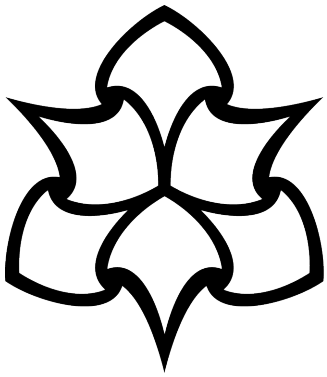


Enhanced Encryption and Fine-Grained Authorization for Database Systems

Walid Rjaibi

PhD 2020

Enhanced Encryption and Fine-Grained Authorization for Database Systems

Walid Rjaibi

A thesis submitted to Manchester Metropolitan University in partial fulfilment for the degree of

Doctor of Philosophy

Faculty of Science and Engineering

Department of Computing and Mathematics

MANCHESTER METROPOLITAN UNIVERSITY

2020

Abstract

The aim of this research is to enhance fine-grained authorization and encryption so that database systems are equipped with the controls necessary to help enterprises adhere to zero-trust security more effectively. For fine-grained authorization, this thesis has extended database systems with three new concepts: Row permissions, column masks and trusted contexts. Row permissions and column masks provide data-centric security so the security policy cannot be bypassed as with database views, for example. They also coexist in harmony with the rest of the database core tenets so that enterprises are not forced to compromise neither security nor database functionality. Trusted contexts provide applications in multitiered environments with a secure and controlled manner to propagate user identities to the database and therefore enable such applications to delegate the security policy to the database system where it is enforced more effectively. Trusted contexts also protect against application bypass so the application credentials cannot be abused to make database changes outside the scope of the application's business logic. For encryption, this thesis has introduced a holistic database encryption solution to address the limitations of traditional database encryption methods. It too coexists in harmony with the rest of the database core tenets so that enterprises are not forced to choose between security and performance as with column encryption, for example. Lastly, row permissions, column masks, trusted contexts and holistic database encryption have all been implemented IBM DB2, where they are relied upon by thousands of organizations from around the world to protect critical data and adhere to zero-trust security more effectively.

Acknowledgements

This thesis would not have been possible without the help, guidance and support from many people. First, I would like to thank my Director of Studies, Dr Mohammad Hammoudeh, for all his help, guidance and support throughout this thesis.

I also want to express my deepest gratitude to my friend and colleague Paul Bird, IBM Senior Technical Staff Member, for the opportunity to drive the database security research in his team as well as for all the guidance, help and support he kindly provided to me throughout this research and during the preparation of this thesis.

I am also grateful to my friends and IBM colleagues Calisto Zuzarte and Mokhtar Kandil for all the time they have spent reviewing and validating design documents, research papers, patent applications and for their kind and valuable feedback throughout the research and during the preparation of this thesis.

My most sincere thanks also go to my IBM colleagues Irene Liu, Greg Stager, Mihai Iacob, Mihai Nicolai, Alex Zhang, Hamdi Roumani, Eric Alton, Harley Boughton, Jerry Kiernan, Tyrone Grandison, Scott Logan and Quentin Presley for all their help and input during the implementation of the concepts introduced by this research.

Last but not least, I want to thank my family for their love, support and for having brought so much joy to my life. And a very special thanks to my children Saif, Safa and Haytham who have filled my life with so much happiness and have given me all the energy I needed to write this thesis.

Table of contents

List of figures

List of tables

List of abbreviations

DBA Database Administrator

SA System Administrator

DBMS Database Management System

RDBMS Relational Database Management System

SQL Structured Query Language

UDF User Defined Function

MQT Materialized Query Table

MAC Mandatory Access Control

LBAC Label-Based Access Control

MLS Multilevel Security

VPD Virtual Private Database

FGAC Fine-Grained Access Control

AES Advanced Encryption Standard

CBC Cipher Block Chaining

IV Initialization Vector

RSA Rivest-Shamir-Adleman

DES Data Encryption Standard

3DES Triple DES

SHA Secure Hash Algorithms

EFS Encrypted File System

SED Self-Encrypting Disk

SSL Secure Socket Layer

TLS Transport Layer Security

HSM Hardware Security Module

LDAP Lightweight Directory Access Protocol

SIEM Security Information and Event Management

KDC Key Distribution Center

XML eXtensible Markup Language

PCI DSS Payment Card Industry Data Security Standard

HIPAA Health Insurance Portability and Accountability Act

GDPR General Data Protection Regulation

Chapter 1: Introduction

This thesis is a PhD by publication. It represents a research journey in database security that has resulted in a portfolio of 8 research papers in peer reviewed journals and conferences, and 7 peer reviewed patents. The thesis highlights the contributions made during the last three years (including the PhD registration period), but also builds upon the author's previous research.

This first chapter briefly introduces database systems, their typical deployment and usage. Next, the key challenges in fine-grained authorization and encryption for database systems are discussed. Then, the chapter presents the motivation, aim and objectives of the research. Lastly, the chapter summarizes the key contributions made in this thesis and gives the outline for the next chapters. Chapter 2 reviews the key tenets of database security, positions the research portfolio in that field and summarizes the key contributions for each. Chapter 3 gives the details of this thesis's contributions to the fine-grained database authorization area. Chapter 4 describes the details of this thesis's contributions to the database encryption area. Chapter 5 describes the details of this thesis's contributions to the mandatory access control area. Chapter 6 shows how the contributions made in this thesis come together to help organizations effectively adhere to zero-trust database security. Chapter 7 summarizes the thesis and explores future directions for database security research.

1.1 Background

Database systems are at the core of an organization's information system. They store critical data such as employee personal data, client transaction data, patient medical records and intellectual property information. Organizations rely upon database systems to ensure the integrity, availability and security of their critical data. They also trust database systems to meet the stringent performance expectations of mission critical applications such as financial transactions and retail sales. Database systems are also relied upon for their compression capabilities to optimize storage costs.

Database systems are usually accessed by two types of personas: A Database Administrator (DBA) and an application user. DBAs are responsible for the installation and ongoing maintenance of the database system software as well as the daily operations such as database backups, restores, configurations and security. Application users access the database through an application, resulting in a multitiered environment where the user's browser is the first tier, the application server is the middle tier and the database server is the third tier. Typically, users log on to the application and the application issues queries to the database to serve the needs of those users. Figure 1.1 depicts a typical database system deployment and usage.

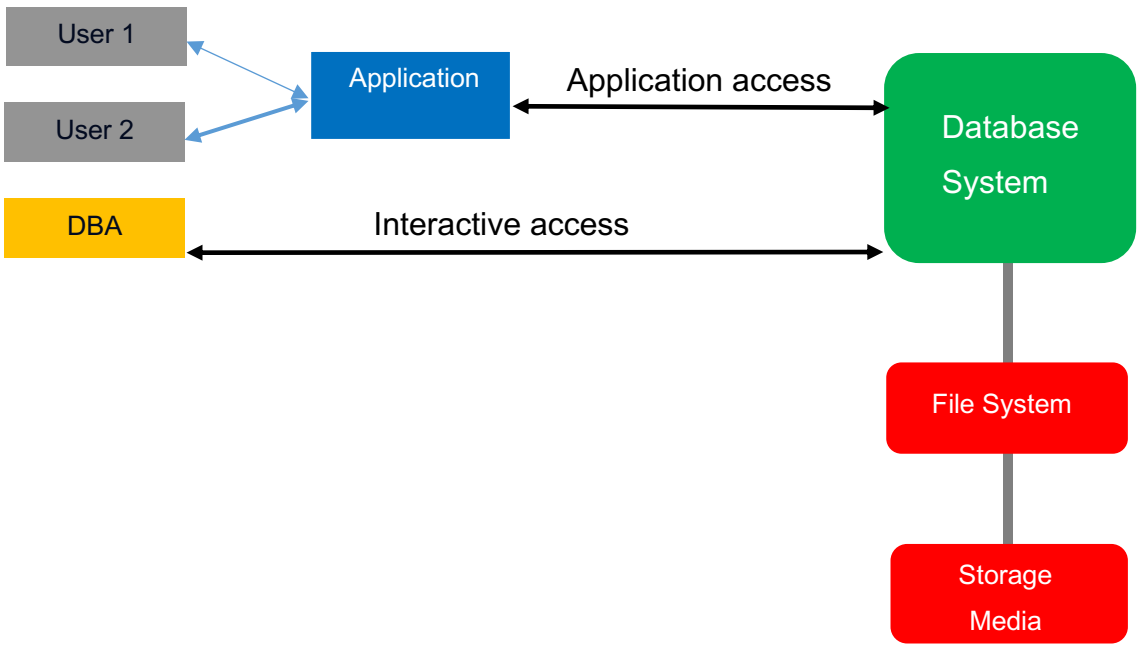


Figure 1.1– Typical database system deployment and usage

A set of fine-grained authorization and data encryption techniques have been proposed to equip database systems with the controls necessary to protect the

critical data entrusted with them. In the next sub-sections, we briefly review these techniques and highlight key challenges.

1.1.1 Fine-Grained Authorization

Fine-grained authorization has originally been tackled through the concept of database views (Elmasri et al., 2010). A DBA would create the desired views over the tables containing the sensitive data and grant access to those views based on “need-to-know”. This approach has two major drawbacks. First, it is not data-centric as the security policy is only enforced when the data is accessed through the views. Users with the right privileges can bypass the security policy by accessing the base tables directly. Secondly, views can very quickly become complex to manage as their number grows in order to satisfy the needs of different user groups. Subsequent work around fine-grained database authorization has addressed these two shortcomings to some degree. However, there are still two major issues. First, the loss of user identity in multitiered environments which renders the fine- grained authorization policies defined in the database almost of no value since the identity of the end user is not known. The application server is then forced to compensate by implementing the fine-grained authorization logic in the application itself. This in turn renders the application more complex, error prone, and prevents it from benefiting from delegating the security policy to the database system where it can be enforced more efficiently. The second major issue is the coexistence of fine-grained authorization with important database tenets such as integrity and performance. How do we balance integrity and security when both a trigger and fine-grained authorization policy are defined on the same table? Similarly, how do we balance performance and security when a Materialized Query Table (MQT) is defined on one or more tables protected with a fine-grained authorization policy? In both cases, enforcing the fine-grained authorization policy blindly can compromise database integrity and/or disrupt the accuracy of a query results set.

1.1.2 Data Encryption

For database encryption, the solutions available in this space can be grouped into four categories: Column encryption, tablespace encryption, file system encryption, and self-encrypting disks. Column encryption negatively affects database performance as queries with range predicates cannot benefit from index-based access plans to limit the data to read from the table. Instead, the

database system is forced to read the entire table to evaluate the query. Tablespace encryption may leave certain data vulnerable to attacks when, for example, a DBA inadvertently takes an action that moves data from an encrypted tablespace to an unencrypted one. File system encryption and self-encrypted disks provide no protection against privileged users on the operating system. As long as the file permissions allow access, such users can easily view the content of the database by browsing the underlying files on the operating system.

1.1.3 Mandatory Access Control

Within the intelligence and defense communities, Mandatory Access Control (MAC) (Rjaibi et al., 2004) is actually the mechanism relied upon for database fine-grained authorization. Under this model, each row in a table is assigned a classification. Similarly, each database user is assigned a clearance. The combination of the MAC rules, the row's classification and the user's clearance determine whether or not a given user can access a given row. MAC solutions for database systems have solely focused on Multilevel Security (MLS) (Rjaibi, 2004). MLS is a very specific MAC model which came out of the US defense community and has rigid classification, clearance and MAC rules. This meant that MLS database systems could not be used to meet the needs of the defense and intelligence communities from other countries where the classification, clearance, and MAC rules may not necessarily match those of the US government. Additionally, the issues around the loss of end user identity in multitiered environments discussed earlier still apply when MAC models are enforced by the database systems.

1.2 Motivation

The rise of data breaches has driven many organizations nowadays to implement zero-trust security in order to reduce the risk of incurring a data breach. Like identity systems and networks (Gilman et al., 2017), database systems also need to evolve to help organizations effectively adhere to zero-trust security for at least three main reasons.

First, database systems store the organization's most critical data (e.g., employee personal data, client transaction data, patient medical records, intellectual property information) and are often the primary target of attacks by malicious entities such as disgruntled employees or external hackers. Second, database systems are the subject of numerous regulations and standards such

as the European General Data Protection Regulation (GDPR) (Voigt et al., 2017) and the Payment Card Industry Data Security Standard (PCI DSS) (Chuvakin et al., 2009) which impose severe financial penalties on any organization that fails to adequately protect critical data. Last but not least, traditional encryption and fine-grained authorization solutions for database systems are not adequate to address the challenges posed by security threats and compliance requirements. As pointed out in Section 1.1, traditional database encryption methods either negatively affect performance (column encryption) or create attack opportunities for malicious users (tablespace encryption, file system encryption, disk encryption). Similarly, traditional fine-grained database authorization methods can be bypassed (e.g. views) and do not address the loss of user identity problem, rendering them unusable in multitiered environments. They additionally do not coexist in harmony with fundamental database tenets such as triggers and MQT, thus creating potential for data leakage. Also, the loss of user identity in multitiered environments diminishes user accountability as auditing at the database level will not be able to show who actually performed which action.

Clearly, traditional database encryption and fine-grained authorization methods are creating a dilemma for enterprises when it comes to meeting their security needs. Some traditional encryption methods provide good security, but that security comes at the expense of database performance. Other encryption methods do not affect database performance, but that advantage comes at the expense of database security. Additionally, traditional fine-grained authorization methods do not apply in multitiered environments, forcing enterprises to build that security in the application. But this renders applications more complex and prevents them from delegating fine-grained authorization to the database where it can be enforced more effectively. The three key research questions are therefore the following:

1. How can database systems be extended to build an encryption solution that meets the security needs but does not come at the expense of core database tenets such as performance and compression?
2. What extensions can be made to database systems to develop a fine-grained authorization solution that enables applications in multitiered environments to delegate the security policy to the database and improves overall database security?

3. How can database systems be extended to build a mandatory access control solution that addresses the limitations of traditional Multilevel Security (MLS) which imposes a rigid security label structure and access rules?

1.3 Aims and Objectives

The aim of this research is to enhance encryption and fine-grained authorization for database systems to help organizations meet their security and compliance needs, without having to compromise any core database tenets such as performance, integrity, compression and without requiring any changes to database applications.

In order to achieve this aim, this research will:

1. Develop a holistic database encryption solution that meets the security needs while coexisting in harmony with core database tenets such as performance and compression.
2. Build a fine-grained authorization solution that enables applications in multitiered environments to delegate the security policy to the database while coexisting in harmony with performance, triggers, User-Defined Functions (UDF) and Materialized Query Tables.
3. Enhance Mandatory Access Control (MAC) in database systems to broaden its applicability to additional use cases such as fine-grained authorization for XML documents stored database tables.
4. Measure the impact of the enhancements introduced on database performance.

1.4 Contributions

This thesis has advanced the areas of database encryption, fine-grained database authorization and mandatory access control. The key contributions can be summarized as follows:

1. A holistic database encryption solution which allows organizations to meet their security and compliance requirements without having to make compromises either on the security side or on the database side

2. A fine-grained database authorization solution which allows organizations to reduce the complexity of their applications and improve overall database security
3. A solution which extends database systems to automatically and transparently enforce privacy policies
4. A multi-purpose mandatory access control solution which addresses the limitations of traditional Multilevel Security (MLS)
5. A fine-grained authorization solution for XML which improves over traditional node-based XML access control approaches, by considering inter-node relationships as the control granularity, and by using the multi-purpose mandatory access control above for controlling access to such inter-node relationships
6. The implementation of the enhancements above in several commercial database systems including IBM DB2 and Informix, where they are relied upon by thousands of clients around the world to protect their critical data and meet their compliance mandates.

1.5 Thesis Organization

The rest of the thesis is organised as follows. Chapter 2 reviews the key tenets of database security, positions the research portfolio in that field and summarizes the key contributions for each. Chapter 3 gives the details of this thesis's contributions to the fine-grained database authorization area. Chapter 4 describes the details of this thesis's contributions to the database encryption area. Chapter 5 describes the details of this thesis's contributions to the mandatory access control area as well as to XML fine-grained authorization. Chapter 6 shows how the contributions above come together to help organizations effectively adhere to zero-trust database security. Chapter 7 summarizes the thesis and explores future directions for database security research. Appendix A, Appendix B and Appendix C list the research portfolio for fine-grained authorization, data encryption and mandatory access control respectively.

Chapter 2: Research Portfolio Overview

This chapter briefly summarizes database security and positions the research portfolio within this field. It then gives a high-level overview of the publications in the portfolio and shows where each fit with respect to the fine-grained authorization, data encryption and mandatory access control areas. Each of these portfolio areas are then discussed in full details in Chapter 3, Chapter 4 and Chapter 5 respectively. The primary focus of this thesis is the portfolio developed during the last 3 years (including the PhD registration period), namely chapters 3, 4 and 6.

2.1 Introduction

Database security is the set of capabilities organizations depend upon to ensure the security of the data they store in databases. It can be broadly divided into five main pillars:

1. **Authentication:** This is the first protection measure where the database system challenges the user to prove who they claim they are. Database systems typically support various options for doing this validation such as verifying the credentials submitted within the database system itself or integrating with an external system to do so. Typical options for an external system include the host operating system, an LDAP server or a Kerberos KDC (MIT, 2019).
2. **Coarse-grained authorization:** This is the next level of protection where the database system verifies that the authenticated user has the privilege to execute a particular action. For example, when a user issues an SQL SELECT statement on given table, the database system must first verify that the user has been granted SELECT privilege on that table. DBAs use the GRANT and REVOKE SQL statements to grant or revoke a particular privilege to/from a user (Elmasri et al., 2010). These privilege assignments are stored in the database system catalog tables and are consulted during authorization checking. Users can acquire a privilege directly or indirectly through membership in a role or group. Memberships in roles and groups are also stored in the database system catalog tables and are consulted during authorization checking.
3. **Fine-grained authorization:** While coarse-grained authorization dictates whether or not a user has the privilege to access a table, fine-grained authorization goes a level deeper. It controls what specific rows, columns or cells of that table the user is allowed to access. Traditionally, database views have been used to enforce fine-grained authorization (Elmasri et al., 2010). A database view represents a dynamically computed set of rows from one or more tables. Typically, the DBA creates the desired views and grants access on those views to the appropriate users. Mandatory Access Control (MAC) is another option some database systems offer for enforcing fine-grained authorization (Rjaibi et al., 2004). It is an option that is typically used by the defense and intelligence communities. In MAC, each data row in a table is assigned a classification representing the

sensitivity of that row (e.g., SECRET). Similarly, users are assigned clearances, defining their access level (e.g., TOP SECRET). The combination of row classification and user clearance determines whether or not the user can access the given row.

4. **Data encryption:** Data encryption can be divided into two categories: Encryption for data in transit and encryption for data at rest. Encryption for data in transit protects the confidentiality of the data exchange between the database system and an application. Most database systems implement Transport Layer Security (TLS) to provide this protection. The goal of encryption for data at rest is to safeguard the data when it is in storage. Different implementations exist ranging from column encryption, to tablespace encryption, to file system encryption, to self-encryption disks.
5. **Auditing:** This is the mechanism database systems provide so enterprises can hold users accountable for their actions. It is also a requirement for complying with various mandates such as the European General Data Protection Regulation (GDPR) or the Payment Card Industry Data Security Standard (PCI DSS). Most database systems provide the flexibility to decide what type of activity to audit such as auditing a specific user, a specific role, a specific table, all users, all tables, and so on. Similarly, most database systems offer several options as to where the audit records are sent. Options include storing them locally on the host operating system or sending them to a Security Information and Event Management (SIEM) system where they are aggregated and correlated with audit data from other applications.

Figure 2.1 highlights the specific database security pillars that are the subject of the research portfolio upon which this thesis is based. The portfolio specifically focuses on fine-grained authorization and data encryption. The rest of this chapter is organized as follows. Section 2.2 summarizes the portfolio of publications related to fine-grained authorization. Section 2.3 highlights the portfolio of publications in the data encryption area. Section 2.4 gives an overview of the portfolio of publications in the area of mandatory access control. Lastly, Section 2.5 concludes this chapter.

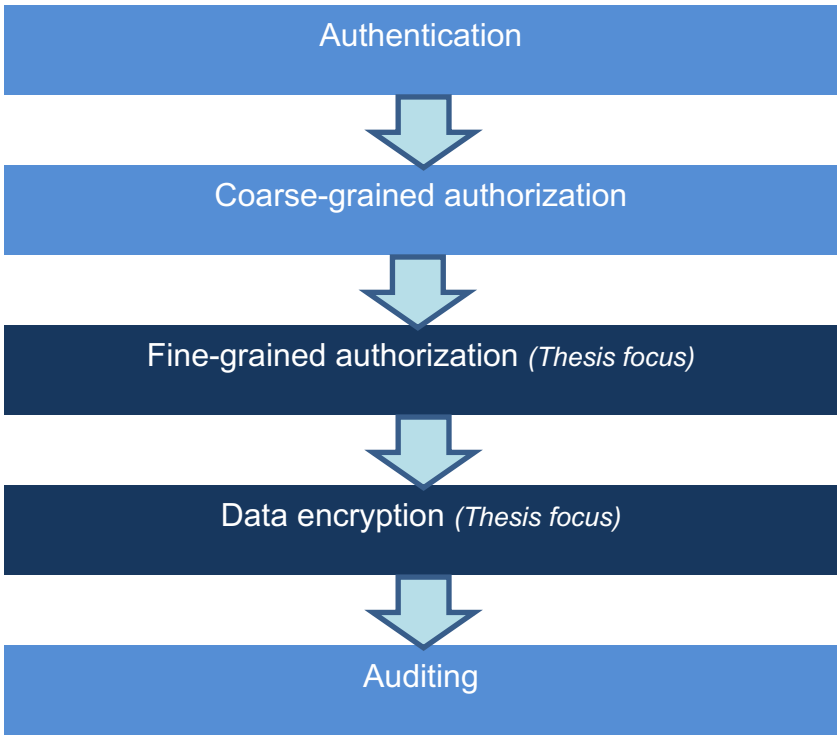


Figure 2.1– Database security pillars and focus of the thesis

2.2 Fine-Grained Authorization

The core element of the research portfolio in this area is the publication "*Enhancing and Simplifying Data Security and Privacy for Multitiered Applications*". This publication was fully developed during the course of this thesis and is given in Chapter 3. It builds upon the ideas expressed in the following patents[1]:

- US Patent US8,234.299B2: "*Method and System for Using Fine-Grained Access Control (FGAC) to Control Access to Data in a Database*". This patent is the foundation for the ***row permission*** and ***column mask*** concepts discussed in the core publication above.
- US Patent US 7,647,626B2: "*Method for Establishing a Trusted Relationship Between a Data Server and a Middleware Server*". This patent is the foundation for the ***trusted context*** concept discussed in the core publication above.

The research portfolio includes additional contributions to fine-grained authorization. Publication "*Extending Relational Database Systems to*

[1] Walid was directly involved in the naming of all 7 patents referred to in this thesis.

Automatically Enforce Privacy Policies" describes a model where privacy policies such as P3P (Agrawal et al., 2005) can be automatically enforced by the database system. This publication builds upon the ideas expressed in the following patents:

- US Patent US7,865,521B2: "*Access Control for Elements in a Database Object*". This patent is the foundation for the ***table restriction*** concept discussed in the publication above.
- US Patent US 7,243,097 B1: "*Extending Relational Database Systems to Automatically Enforce Privacy Policies*". This patent is the foundation for the method to translate privacy policies into table restrictions discussed in the publication above.

Table 2.1 summarizes the fine-grained authorization publications and their key contributions.

Table 2.1 – Fine-grained authorization publications

ID	Publication	Key Contributions
1	Rjaibi, W., Hammoudeh, M. (2020). '**Enhancing and Simplifying Data Security and Privacy for Multitiered Applications'**. *Journal of Parallel and Distributed Computing, Special Issue on Enabling Technologies for Energy Cloud.* (Also, Chapter 3 of this thesis) Walid's % contribution: 75.	- Design of a holistic fine-grained database authorization solution which allows organizations to reduce the complexity of their applications and improve overall database security. - Enable organizations to adhere to zero-trust security. - Implementation of the solution in IBM DB2 for Linux, Unix and Windows, IBM DB2 for z/OS and IBM for DB2 for iSeries.
2	**Method and System for Using Fine-Grained Access Control (FGAC) to Control Access to Data in a Database** *US Patent US8,234.299B2* Walid's % contribution: 50.	This patent is the foundation for the row permission and column mask concepts discussed in the core publication #1 above.
3	**Method for Establishing a Trusted Relationship Between a Data Server and a Middleware Server** *US Patent US 7,647,626B2* Walid's % contribution: 50.	This patent is the foundation for the trusted context concept discussed in the core publication #1 above.
4	Agrawal, R., Bird, P., Grandison, T., Kiernan, J., Logan S., Rjaibi, W. (2005). 'Extending relational database systems to automatically enforce privacy policies'. *In Proceedings of the International Conference on Data Engineering (ICDE).* Walid's % contribution: 50.	- Design of a solution which extends database systems to be able to automatically enforce privacy policies. - Enable organizations to meet privacy requirements for data stored in database systems.
5	**Access Control for Elements in a Database Object** *US Patent US7,865,521B2* Walid's % contribution: 50.	This patent is the foundation for the table restriction concept discussed in publication #4 above.
6	**Extending Relational Database Systems to Automatically Enforce Privacy Policies** *US Patent US 7,243,097 B1* Walid's % contribution: 50.	This patent is the foundation for the method to translate privacy policies into table restrictions discussed in publication #4 above.

2.3 Data Encryption

The core element of the research portfolio in this area is the publication "*Holistic Database Encryption*". A summary of this publication is given in Chapter 4 and the publication itself is given in Appendix B.

Publications "*Towards Zero-Trust Database Security – Part 1*" and "*Towards Zero-Trust Database Security – Part 2*" show how the solution discussed in publication "*Holistic Database Encryption*" contributes to implementing zero-trust security for database systems. These two additional publications have been fully developed during the course of this thesis and are the foundation for Chapter 6 (Towards Zero-Trust Database Security). The publications themselves are given in Appendix B.

Table 2.2 summarizes the data encryption publications and their key contributions.

Table 2.2 – Data encryption publications

ID	Publication	Key Contributions
1	Rjaibi, W. (2018). '**Holistic Database Encryption**'. *In Proceedings of the International Conference on Security and Cryptography (SECRYPT).* Walid's % contribution: 100.	- Design of a holistic database encryption solution which allows organizations to meet their security and compliance requirements without having to make compromises either on the security side or on the database side. - Enable organizations to adhere to zero-trust security. - Implementation of the solution in IBM DB2 for Linux, Unix and Windows.
2	Rjaibi, W., Hammoudeh, M. (2019). '**Towards Zero-Trust Database Security Part 1**'. *IEEE Future Directions Newsletter: Technology Policy & Ethics*, Issue (September 2019). Walid's % contribution: 80.	- Introduces a database threat model and raises awareness of the direct and indirect means through which the same data in a database can be accessed.
3	Rjaibi, W., Hammoudeh, M. (2019). '**Towards Zero-Trust Database Security Part 2**'. *IEEE Future Directions Newsletter: Technology Policy & Ethics*, Issue (December 2019). Walid's % contribution: 80.	- Outlines solutions (including encryption) to address the direct and indirect access challenges and to enable zero-trust database security.

2.4 Mandatory Access Control

The core element of the research portfolio in this area is the publication "*A Multi-Purpose Implementation of Mandatory Access Control in Relational Database Management Systems*". A summary of this publication is given in Chapter 5 and the publication itself is given in Appendix C. It builds upon the ideas expressed in the following patents:

- US Patent US7,568,235B2: "*Controlling Data Access Using Security Label Components*". This patent is the foundation for the ***security label*** concept discussed in the core publication above.
- US Patent US7,860,875B2: "*Method for Modifying a Query by Use of an External System for Managing Assignments of User and Data Classifications*". This patent is the foundation for the enterprise integration methodology discussed in the core publication above.

Publication "*Inter-Node Relationship Labelling: A Fine-Grained XML Access Control Implementation Using Generic Security Labels*" shows an application of the multi-purpose MAC solution discussed in the core publication. The fine-grained XML access control solution devised improves over traditional node-based XML access control approaches, by considering inter-node relationships as the control granularity, and by using ***security labels*** to control access to such inter-node relationships. A summary of this publication is given in Chapter 5 and the publication itself is given in Appendix C. This publication builds upon the ideas expressed in the following patents:

- US Patent US2009/0063951A1: "*Fine-Grained, Label-Based, XML Access Control Model*". This patent is the foundation for the inter-node relationship labelling concept discussed in the publication above.

Lastly, publication "*An Introduction to Multilevel Secure Relational Database Management Systems*" surveys and critiques traditional implementation of mandatory access control in database systems (i.e., MLS). This publication is also given in Appendix C.

Table 2.3 summarizes the mandatory access control publications and their key contributions.

Table 2.3 – Mandatory access control publications

ID	Publication	Key Contributions
1	Rjaibi, W., Bird, P. (2004). '**A Multi-Purpose Implementation of Mandatory Access Control in Relational Database Management Systems**'. *In Proceedings of the International Conference on Very Large Data Bases (VLDB).* Walid's % contribution: 75.	- Design of a mandatory access control solution for database systems which addresses the limitations of traditional Multilevel Security (MLS). - Enable organizations to adhere to zero-trust security. - Implementation of the solution in IBM DB2 for Linux, Unix and Windows, and Informix.
2	**Controlling Data Access Using Security Label Components** *US Patent US7,568,235B2* Walid's % contribution: 75.	This patent is the foundation for the security label concepts discussed in the core publication #1 above.
3	**Method for Modifying a Query by Use of an External System for Managing Assignments of User and Data Classifications** *US Patent US7,860,875B2* Walid's % contribution: 50.	This patent is the foundation for the enterprise integration methodology discussed in the core publication #1 above.
4	Zhang, Z., Rjaibi, W. (2006). 'Inter-node Relationship Labelling: **A Fine-Grained XML Access Control Implementation Using Generic Security Labels**'. *In Proceedings of the International Conference on Security and Cryptography (SECRYPT).* Walid's % contribution: 50.	- Design of a solution which improves over traditional node-based XML access control approaches, by considering inter-node relationships as the control granularity. - Enable databases to extend fine-grained authorizations to XML columns in database tables. - Enable organizations to meet privacy requirements and adhere to zero-trust security.
5	**Fine-Grained, Label-Based, XML Access Control Model** *US Patent US2009/0063951A1* Walid's % contribution: 50.	This patent is the foundation for the inter-node relationship labelling concept discussed in the publication #4 above.
6	Rjaibi, W. (2004). '**An introduction to multilevel secure relational database management systems**'. *In Proceedings of the conference of the Centre for Advanced Studies on Collaborative research (CASCON).* Walid's % contribution: 100.	Survey and critique of traditional implementations of mandatory access control in database systems (i.e., MLS).

2.5 Conclusion

This chapter has positioned the research portfolio within the database security field. It has also given a high-level overview of the publications in this portfolio and shows where each fit with respect to the fine-grained authorization, data encryption and mandatory access control areas. The next three chapters will discuss this research portfolio in full details. The publications themselves are given in Appendixes A, B and C.

Chapter 3: Enhanced Fine-Grained Authorization

This chapter highlights the shortcomings of traditional fine-grained authorization approaches in database systems, in particular the loss of user identity in multitiered application environments which prevents such applications from delegating the security policy to the database where it can be enforced more effectively. Next, the chapter introduces the ***row permission***, ***column mask*** and ***trusted context*** concepts to extend database systems so that applications can safely delegate the security policy to the database as opposed to building such policy in the application logic itself. The implementation of such concepts in IBM DB2 is then discussed and a performance evaluation is presented. The evaluation shows that enforcing the fine-grained database authorization policy by the database has not resulted in any significant performance drawbacks for the application. This means that the gains in security and the reduction in application complexity do not come at the expense of database performance. This chapter appears in the research portfolio as publication "*Enhancing and Simplifying Data Security and Privacy for Multitiered Applications*", which was fully developed during the course of the PhD registration. It is a synthesis of the entire research portfolio in fine-grained authorization for database systems.

3.1 Introduction

Classical 3-tier applications have become quite complex partly due to the cost of implementing data security and privacy rules within the application logic itself. Figure 3.1 shows the architecture of a classical 3-tier application, where the end user browsers, the application server and the database server represent the first, second and third tier respectively.

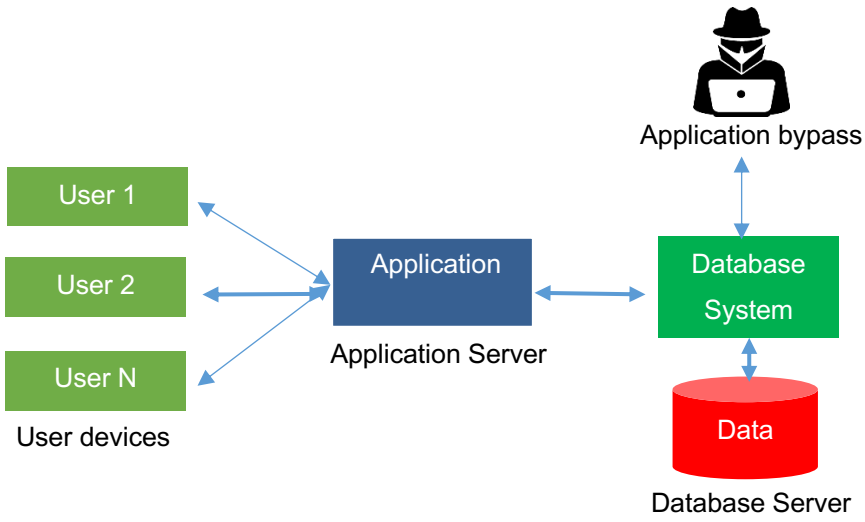


Figure 3.1– Classical 3-tier application architecture

Under this model, end users access the application to perform tasks related to their job. The application authenticates such users to ensure they are authorized to use the application. To meet the needs of the end users, the application makes a connection to the database using a generic user ID identifying that application to the database. To ensure that the right content of the database is returned to the right users, the application logic typically includes a fine-grained authorization layer to do the appropriate level of data filtering. This layer is usually implemented in one or a combination of these two options:

- The application builds the SQL queries in such a way that they include the appropriate predicates and functions to filter out and mask the table data as appropriate.
- The application builds a set of database views which perform the appropriate level of data filtering and routes the SQL queries to the appropriate views based on user identities.

Besides burdening the application with the task of implementing fine-grained authorization, this model also suffers from other security drawbacks including:

- The approach is not data-centric. This means that the intended security policy is not enforced when the application is bypassed. An example of such bypass is when the application administrator chooses to abuse the application's database user ID to access the database directly. This is particularly important in today's world where internal threats are as concerning as external threats (Zaytsev et al., 2017), (Ghafir et al., 2018).
- Over granting of database privileges. The application's database user ID is typically granted the privileges of a database administrator so that it can be used to do all things on behalf of all users. This means that when such user ID is abused, the consequences to the organization can be severe.
- Loss of end user identity at the database level. This is a consequence of the application doing all database accesses on behalf of all users using a single user ID. This makes it impossible to leverage database auditing to hold end users accountable for their actions. It also prevents the application from delegating the fine-grained authorization policy to the database as the user ID is lost at that level.
- Unnecessary exposure of the security policy to application developers.

We contend that applications complexity can be reduced by delegating the fine-grained authorization task to the database system. We also contend that this delegation will additionally address the security concerns raised above and enable applications to better adhere to compliance mandates such as the European General Data Protection Regulation (GDPR) (Voigt et al., 2017) and the Payment Card Industry Data Security Standard (PCI DSS) (Chuvakin et al., 2009).

The crux of our contribution is the design of a holistic fine-grained database authorization approach which allows organizations to reduce the complexity of their applications and improve overall database security. We have also implemented the solution in a commercial database system (IBM DB2 for Linux, Unix, and Windows). Our approach improves over the state of the art as follows:

- Fine-grained authorization coexists in harmony with fundamental database tenets such as performance and integrity so that organizations are not forced to make compromises either on the security side or on the database side.

- Applications can safely delegate the security policy to the database system by leveraging the trusted context concept to propagate user identities to the database system, thus extending the value of fine-grained database authorization to multitiered applications.
- Organizations can leverage the trusted context concept to ensure that the application's database user ID cannot be abused by malicious entities who may want to leverage that user ID for accessing the database outside the scope of the application (i.e., application bypass).

The rest of this chapter is organized as follows. Section 3.2 reviews the related work. Section 3.3 describes our fine-grained database authorization model. Section 3.4 introduces our trusted context concept which addresses the loss of user identity problem in multitiered environments. In section 3.5, we discuss how the new concepts introduced safely coexist with core database tenets. Section 3.6 describes the performance evaluation of our fine-grained database authorization model. In Section 3.7, we discuss a banking use case and show how our solution meets its requirements. Lastly, Section 3.8 summarizes this chapter.

3.2 Related Work

Traditionally, fine-grained authorization in database systems has been implemented using the concept of database views (Elmasri et al., 2010). Like database views, our approach is an extension to SQL and is declarative in nature. Administrators are not expected to write any code to implement the fine-grained authorization rules. However, our solution improves over database views in two main ways. First, our approach defines the row and column controls directly on the database tables themselves. This means that the row and column authorization is always enforced regardless of whether the table is accessed directly or indirectly through a database view. In contrast, when implementing fine-grained authorization using views, the row and column authorization is enforced only when the access is made through those views. In other words, views do not provide any protection when the underlying tables are accessed directly. Additionally, our approach introduced the notion of trusted context to enable user identity propagation in multitiered environments so that applications can safely delegate fine-grained authorization to the database system.

Oracle Virtual Private Database (VPD) was, to the best of our knowledge, the first database system to introduce a fine-grained authorization model that improves over traditional database views (Gaetjen et al., 2015) and is the closest to our work. There are however some important differences between Oracle VPD and our approach. First, the Oracle VPD approach is not declarative. It requires the administrator to code a PL/SQL program which computes a predicate string that is appended to any SQL statement accessing the table with which the PL/SQL program was associated. This also limits the benefits of SQL statements caching only to situations where the PL/SQL program is guaranteed to return the same results for all users. Our approach does not limit the benefits of SQL statements caching because it does not change the SQL statement text itself. Oracle VPD also includes the notion of an Application Context which can be used by applications to pass information to the database system such as a user ID in a multitiered environment. An Application Context is a set of name-value pairs the Oracle database systems stores in memory. Our trusted context concept provides a more robust framework for propagating user identities in multitiered environments as it first requires the establishment of a trusted relationship between the database system and the application before propagating a user ID is allowed. It also provides more control on which specific user IDs are allowed for propagation as well as the ability to associate the application's privileges with the trusted context only so they cannot be abused elsewhere.

The Row Level Security (RLS) and Dynamic Data Masking (DDM) capabilities in Microsoft SQL Server are conceptually similar to our row permission and column mask concepts (Carter, 2018). But there are some important differences between the two approaches. First, the SQL Server DDM is static in the sense that the user either has access to the actual value in the column or a masked value thereof. The column mask concept in our approach is dynamic in the sense that the decision of whether the user sees the actual value, or a masked value is determined dynamically based on the conditions expressed in the column mask definition. Additionally, the SQL Server RLS requires the administrator to go through a two-step process: They first need to create a function which returns a filtering predicate, and then create a policy on the table to apply that predicate. In our approach, this is all done in a single step using the row permission concept. The user identity propagation in multitiered environments is supported through an application context concept similar to the Oracle VPD one discussed above.

The Vertica Row Access Policy and Column Access Policy concepts enable administrators to enforce access to table data at the row and column level respectively (Vertica, 2019). The Vertica SQL syntax is very similar to ours. However, and to the best of our knowledge, the Vertica solution does not discuss how it enables user identity propagation in multitiered environments. Additionally, the Vertica solution does not show any performance evaluation to contrast implementing the fine-grained authorization rules within the database versus within the application.

The Sybase Row Level Access Control (RLAC) enables administrators to restrict access to data rows in a table by defining an access rule and binding it to a specific column of the table (Garbus, 2015). When a table is accessed, the access rules in place are automatically enforced by incorporating them into the query at compilation time. Our approach differs from the Sybase RLAC capability in several ways. First, RLAC is limited to row level access control only while our approach covers both the row and column level. Also, to the best our knowledge, the Sybase RLAC does not discuss how it enables user identity propagation in multitiered environments.

The fine-grained authorization model presented in (Chaudhuri et al., 2007) is also a declarative SQL model like ours. But there are some differences between the two approaches. The first difference is fairly minor. They have extended the GRANT SQL statement to give administrators the tools to define row and column authorization rules while our approach introduced these constructs independently of the GRANT statement. However, the work presented in (Chaudhuri et al., 2007) did not cover user identity propagation in multitiered environments. It assumed it was taken care of through a method similar to the application context concept in Oracle VPD. Lastly, their work did not include any performance evaluation to contrast implementing the fine-grained authorization rules within the database versus within the application.

The fine-grained authorization approach discussed in (Agrawal et al., 2005) is also a declarative SQL model but there are some key differences with our approach. First, the focus of the work in (Agrawal et al., 2005) is on privacy policies. They introduced row and column restriction concepts for the purpose of being able to map privacy policies to them so the database system can automatically enforce privacy policies. It did not cover user identity propagation

in multitiered environment. Also, the model described in (Agrawal et al., 2005) did not include any performance evaluation to contrast enforcing the privacy policy within the database versus within the application.

The model described in (Rjaibi et al., 2004) can be regarded as a special form of fine-grained authorization. The focus of this work is more around introducing a flexible mandatory access control model which addresses some of the shortcoming of classical Multilevel Security (Rjaibi, 2004). It is a declarative SQL model and also ensures the security predicates are executed before any potentially unsafe predicates to prevent data leakage. However, it did not introduce the concept of secure functions as we did in this chapter, so security predicates are always executed first even if that does not make sense from a performance perspective. Lastly, the approach discussed in (Rjaibi et al., 2004) did not cover user identity propagation in multitiered environments.

Besides security built into database systems themselves, the importance of protecting databases has also led to the emergence of external database security tools. The leading tools in this context are Guardium (Chen et al., 2014) and Imperva (Imperva, 2019). These tools can be thought of as complementary to our solution as they focus more on database auditing, compliance reporting and analytics on auditing data as opposed to fine-grained database authorization.

3.3 Fine-Grained Database Authorization Model

We extend the SQL table privileges model with two new concepts: *Row permissions* and *column masks*. Row permissions and column masks implement a second layer of security on top of table privileges. When a table is accessed, the privileges layer determines whether or not the table can be accessed. Next, row permissions are applied to decide what specific set of the table rows the user is authorized to access. Lastly, column masks are applied to figure out whether the user is allowed to see the actual value in a column or a masked value thereof. For example, row permissions ensure that when a doctor queries the patients table, they only see rows that represent patients under their care. On the other hand, a column mask on the phone number column ensures that the doctor sees only phone numbers for patients who consented to share their phone numbers with them. Figure 3.2 shows our model as an extension to the SQL compiler.

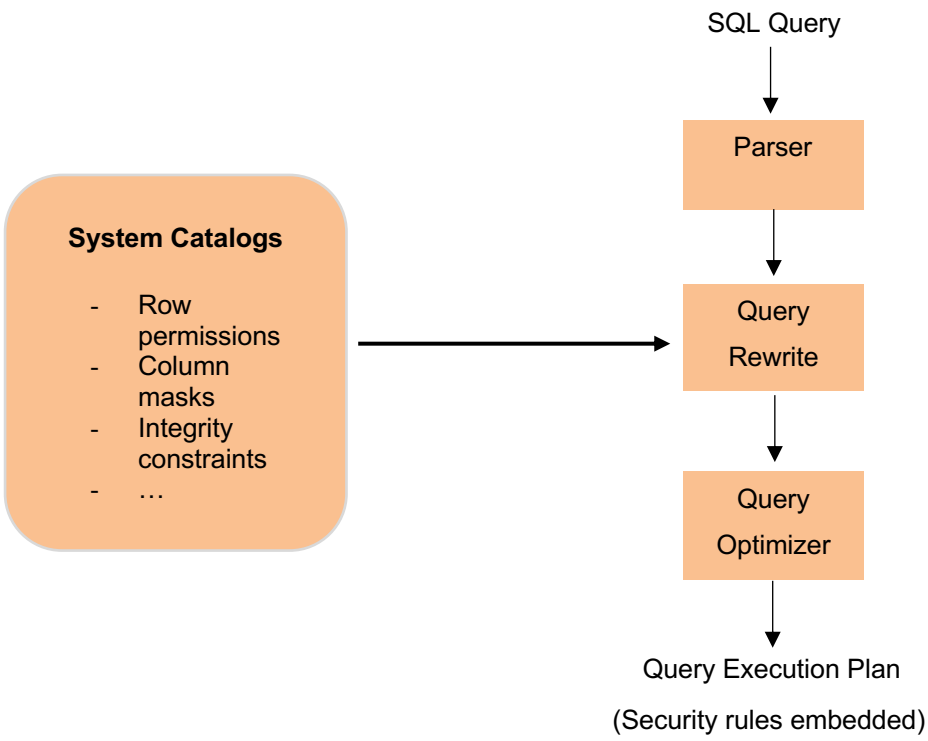


Figure 3.2– Fine-grained authorization as an extension of the SQL Compiler

An SQL statement first goes through the parser component where it is analyzed for syntactic correctness and a query graph is generated. Next, it goes into the query rewrite component where the graph is modified to inject additional objects such as integrity constraints and triggers. We have modified this component to inject the new row permission and column mask concepts we have introduced. The modified graph then goes into the query optimizer component where several execution options are examined, and the optimal plan is selected based on a cost function. We have also modified this component to protect against potential data leakage should an unsafe predicate be evaluated before the security rules expressed by the row permissions are evaluated.

Unlike database views (Elmasri et al., 2010) where the security policy is enforced only when the views themselves are accessed, row permissions and column masks are table centric. This ensures that the security policy is enforced consistently regardless of how the table is accessed. Row permissions and column masks are also applied uniformly across all users, including DBAs, which helps organizations better adhere to zero-trust security (Gilman et al., 2017), (Walker-Roberts et al., 2018), (Hammoudeh et al., 2018) and in particular ensuring that access control is based on "*need-to-know*". Additionally, row

permissions and column masks are application transparent. Database applications can immediately benefit from these concepts without having to incur any code changes. The SQL syntax for row permissions and column masks is given below.

```
create permission permission-name on table-x
        for rows where predicate-clause
        enforced for all access [disable | enable]

create mask mask-name on table-x
        for column column-name
        return case-expression [disable | enable]
```

Example 1

The following row permission creates a rule that grants access to rows in the PAYROLL table only to users who are members of the HR role.

```
create permission rpayroll on payroll
        for rows where verify_role_for_user (USER, 'HR') = 1
        enforced for all access enable;
```

Example 2

The following column mask creates a rule that grants access to the salary column in the PAYROLL table only to users who are members of the SM role. Other users will see NULL when they query the salary column.

```
create mask msalary on payroll
        for column salary
        return case when verify_role_for_user (USER, 'SM') = 1
                    then salary
                    else null
                end
         enable;
```

Some applications may not desire receiving a NULL value. Instead, they may want to receive an alternate and format preserving data value (Goldsteen et al., 2015). Our model can easily support this use case. All that is needed is to register a User Defined Function (UDF) in the database and modify the CREATE MASK

SQL statement above such that instead of returning NULL, call the UDF to return the desired output.

A table can have zero or more row permissions. When more than a single row permission is defined on a table, the predicates from each one of them are combined together by applying the logical OR operator. In other words, if a row permission R_1 gives user U_1 access to a set of rows S_1, and another row permission R_2 on the same table gives that same user access to another set of rows S_2, then both row permissions would give that user access to the union of S_1 and S_2. A column can have zero or one mask. We extended the SQL compiler so that during query compilation, row permissions and column masks are dynamically injected into the query graph. This ensures that the query execution plan generated automatically enforces the rules expressed by the row permissions and column masks.

3.3.1 Row Permissions Enforcement

Row permissions defined on a given table are automatically applied when that table is accessed through any table level SQL statements: SELECT, INSERT, UPDATE, DELETE, and MERGE.

For SELECT statements, the predicates from all the row permissions defined on the table are combined together through the logical OR operator to derive a master predicate. This master predicate acts as a filter to limit the set of rows returned. We extended the query optimizer component of the SQL compiler to ensure that this master predicate is evaluated before any other unsafe user predicates. This is important to guard against potential data leakage through such unsafe user predicates. For example, suppose there is a UDF which emails the table rows retrieved to some external party. If such UDF appears in a user predicate and that predicate is executed before the master predicate, then by the time the master predicate is applied it will already be too late as the row would have already been sent out.

For INSERT statements, the rules specified in the row permissions defined on that table are used to determine whether or not the row can be inserted into the table. To qualify, the user attempting to insert the row must be able to retrieve it back through a SELECT statement. This semantic is analogous to how symmetric

database views behave. More specifically, a user is not allowed to insert a row they cannot retrieve back.

For UPDATE statements, the rules specified in the row permissions defined on that table are used to determine whether or not the row can be updated. This is a two-step process. First, the row permissions are used to filter out the set of rows that can be updated. In other words, a user cannot update rows they are not allowed to see. Next, the updated rows (if any) must conform to the same semantic as for INSERT processing to ensure that the user does not inject rows they cannot retrieve back.

For DELETE statements, the rules specified in the row permissions defined on that table are used to filter the set of rows that can be deleted in order to ensure that the user can only delete rows they can see.

A MERGE statement can be thought of as a combination of an INSERT and an UPDATE statements. Therefore, a MERGE statement is processed as an INSERT when dealing with new rows and as an UPDATE when dealing with existing rows in the table.

3.3.2 Column Masks Enforcement

The goal of a column mask defined on a given column C1 is to ensure that when C1 appears in the final results set of a query, C1 values are masked out if the user is not authorized to see them. This has two important implications. First, the SQL compiler will enforce the column mask for SELECT statements only. INSERT, UPDATE, DELETE, and MERGE statements do not return a result set to the user, so the column mask does not apply in these cases. Secondly, the SQL compiler must ensure that the enforcement of a column mask does not break database applications as this can have severe business impact. For example, suppose that a column mask is applied when the column appears in a predicate. This may totally change the final results set and the database application may end up processing a different set of rows (e.g. giving a raise to the wrong employees). Consequently, we have extended the SQL compiler such that column masks do not interfere with the computation of the final results set and the order or grouping thereof. More specifically, column masks are not applied when the column appears in any of these situations: WHERE clauses, GROUP BY clauses, HAVING clauses, SELECT DISTINCT, and ORDER BY clauses.

One consequence of this approach is that it may create opportunities for inferences. But as discussed in Section 1, we focus on application access as opposed to free direct SQL access to the database. Furthermore, the trusted context concept introduced later in this chapter enables establishing a trusted relationship between the application and the database server as well as protecting against abuse of the application's database user ID.

3.4 User Identity Propagation in Multitiered Environments

In multitiered environments, the middle tier application serves the needs of several users over a pooled database connection. Under this model, the database server only sees a generic user ID which identifies the middle tier application, not the actual users of that application. Despite being a very popular application model, the fact that the database server only sees a generic user ID for all accesses poses several challenges.

First, the middle tier application cannot benefit from fine-grained database authorization because the database server does not see the identity of the application user. Thus, instead of delegating the authorization burden to the database server where it can be enforced more effectively, the middle tier application is forced to implement that fine-grained authorization in the application itself. This renders the application more complex, exposes the security policy to application programmers, and forces unnecessary patching of the application each time the security policy needs to be updated.

Additionally, using a single user ID for all database accesses diminishes user accountability. For example, one of the very first tasks in a forensic investigation is to check the database audit logs for gaining insight into user activities. However, if all accesses by all users are made using a single user ID, the database audit log would unfortunately provide little to no value.

The naïve approach to address this issue is to have the middle tier application establish a separate database connection for each user. Unfortunately, this approach may not be always feasible as the middle tier application may not have access to the end user database credentials. Additionally, even if this were feasible, this approach would not be desirable as establishing a large set of database connections would introduce a database performance overhead. This

is the overhead associated with user authentication and the setting of the actual connection structures on the database server side.

Clearly, a better approach is needed for relieving the middle tier application from the burden of enforcing fine-grained authorization, and for holding users accountable for their actions.

3.4.1 Trusted Contexts

We extend database systems by introducing a new concept called ***trusted context***. A trusted context is a database object which defines a trust relationship between the database server and an external entity such as a middle tier application server. The trust relationship allows the database security administrator (DBSECADM) to specify a set of conditions which, when satisfied by a database connection request, instructs the database server to internally mark that database connection as trusted. A trusted connection gives the entity that established such connection a set of privileges that are not available outside the scope of that trusted connection. One example of such privileges is the ability to reuse an existing database connection for a different user without having to re-authenticate that user at the database server. Reusing an existing database connection avoids incurring a performance overhead by eliminating the need to establish a new database connection. Therefore, a middle tier application server can take advantage of the trusted context concept to establish an initial trusted connection, and then reuse that trusted connection to propagate an end user identity to the database server before submitting database requests on behalf of that end user.

The DBSECADM can choose from a variety of attributes to set the conditions for a trusted relationship such as a user ID, an IP address, a domain name, a digital certificate, and the type of encryption used to protect the communication channel between the database server and the middle tier application (e.g., SSL). The SQL language syntax for our trusted context concept is given below.

```
create trusted context context-name
        based upon connection using system authid authorization-id
        attributes key-value-pair-list
        default role role-name
        with use for user | role | group name [without authentication |
        with authentication] [role role-name]
        [disable | enable]
```

Example 3

The following trusted context establishes a trusted relationship between the database server and a middle tier application. The attributes upon which this trusted relationship is based are the user ID identifying the middle tier application itself, the IP address of the server where that application is hosted, and the type of communication encryption used to protect the communication channel between the database server and the middle tier application.

```
create trusted context ctx1
        based upon connection using system authid midtierApp1
        attributes (address '174.94.142.56' encryption 'SSL')
        with use for role midtierApp1Users
        without authentication
        enable;
```

In our implementation of trusted contexts in IBM DB2, we have extended the database server connection processing as follows. When a database connection request is received, we go through the authentication process as usual, but we also compare the attributes of that request with the attributes of the trusted context objects defined at that database server. If there is a match, we mark that connection as trusted. We have also extended the DB2 Command Level Interface (CLI) with a new command to give applications the option to request switching the current user ID on a trusted database connection. On the database server side, when such request is received, we first verify this is within the scope of a trusted connection, and then ensure that the user ID to switch to is authorized as per the trusted context object definition. For example, the trusted context definition above states that it is only permitted to switch to users who are members of the role *midtierApp1Users*. Lastly, we also check whether the trusted context definition authorizes switching users without authentication or requires authentication. If authentication is not required as in Example 3 above, then no further processing is required. Otherwise, the switch user request must provide a valid authentication credential. Once the checks above are completed and the switch user request is authorized, we reset the user environment over the current physical connection to match the new user, and the application is now ready to start sending database commands under the scope of this new user.

Also, in order to ensure database integrity is not compromised, we extended the database server processing such that switching users over a trusted connection is permitted only on transaction boundary. If such a request is made outside of a transaction boundary, the current transaction is rolled back, and the connection is put in an unconnected state, thus giving the middle tier application the opportunity to recover.

3.4.2 Trusted Context-Based Authorization

Traditionally, database security models are such that the privileges granted to a user are universally applicable irrespective of any context. For example, if a user is granted SELECT privilege on the payroll database table, that user could exercise that privilege regardless of how they gain access to the database. The lack of control on when a privilege is available to a user can weaken overall security since the privilege may be abused. For example, an application administrator may choose to use the application's database credentials to connect to the database directly and make changes that are contrary to the application business logic.

To provide control over when privileges may be exercised, we extend the trusted context concept so that a DBSECADM can associate one or more roles with a trusted context. Roles that are associated with a trusted context are only exercisable when the user is acting within the scope of a trusted connection based upon that trusted context. This enables organizations to better adhere to zero-trust security, and in particular the "verify and never trust" tenet as the database system verifies more security attributes before granting a role to user (Gilman et al., 2017), (Walker-Roberts et al., 2018).

Example 4

The definition of the following trusted context is similar to Example 3, but it specifies two database roles. The first role is *DBCONNECT* which the DBSECADM decided not to grant to the user ID *midtierApp1*. Instead, they assigned it to this trusted context. This means that if the application administrator were to abuse this user ID by attempting to connect to the database from a server other than what is stated in the trusted context definition, that connection will be refused by the database server. The second role is HR, which is the role that grants access to the content of the payroll table as per the row authorization in

Example 1. This in turn means that members of the HR role will have access to the payroll table only within the scope of the trusted connection based upon this trusted context. In other words, they will only have access when they are using the application and not otherwise.

```
create trusted context ctx1
        based upon connection using system authid midtierApp1
        attributes (address 'srv.dep.org.com' encryption 'SSL')
        default role DBCONNECT
        with use for role midtierApp1Users
        without authentication HR
        enable;
```

In our implementation of trusted context-based authorization in IBM DB2, we have extended the database server authorization model as follows. When a database connection request is matched with a trusted context object, we check if there are any default roles assigned to that trusted context and add them to the user's roles list so they are used when deciding whether or not the user is authorized to connect to the database. Similarly, when a request to switch the current user on a trusted connection is received, we check if the trusted context definition grants any roles to the user to switch to and add any such roles to the new user's roles list accordingly.

3.5 Safe Coexistence with Fundamental Database Tenets

Database security needs to safely coexist with fundamental database tenets. Failure to do so may create database vulnerabilities and limit adoption of the solution.

3.5.1 User Defined Functions

A User Defined Function (UDF) is an important database concept which applications depend upon to delegate certain tasks to the database system. We extended the database system such that, by default, the row permission predicates are evaluated first to avoid potential data leakage through UDFs that may also appear in the set of predicates to apply on the table. The following experiment illustrates this extension and can be consistently repeated on any recent IBM DB2 system. The experiment creates a table T1 with 2 integer columns A and B. It inserts 3 rows into this table (1,1), (2,2) and (3,3). Then, we create a UDF which replaces any value in column A that is greater than 1 by 1.

When we run the simple SQL query SELECT A, B FROM T1 WHERE F1(A) = 1, we expectedly obtain 3 rows because the values 2 and 3 in column A are changed to 1 by the UDF F1. Then we create a row permission with the predicate “A = 1”. Now, when we run the SELECT query above any number of times, we consistently get back a single row. This is because our design ensures that the row permission predicates are executed before any unsafe UDF predicate. This is how data leakage is prevented because the UDF could have done anything with the data rows such as modifying them to alter the results set (as F1 does). But our design ensures that the UDF only sees the rows which are authorized for the user running the SELECT query. Below are the exact steps.

```
create table T1 (A int, B int);
insert into T1 values (1,1), (2,2), (3,3);
create function F1 (A int) returns int
        language SQL contains SQL no external action deterministic
        return (case when A > 1 then 1 else A end);
select A, B from T1 where F1(A) = 1;
create permission P1 on T1
       for rows where A = 1
       enforced for all access
      enable;
select A, B from T1 where F1(A) = 1;
```

While executing the UDF predicate last is good from a security perspective, it may not be necessarily good from a performance perspective, particularly if the UDF is a trusted function. Therefore, we extended the database system with the concept of secure UDF. By default, a UDF is not secure, but the administrator can alter the definition of a UDF to mark it secure. This means that the administrator confirms that the UDF is trusted. When a UDF is secure, the database system can order the evaluation of predicates based on such UDF anywhere the SQL compiler sees fit. Secure UDF enable performance and database security to coexist in harmony.

3.5.2 Materialized Query Tables

A Materialized Query Table (MQT) is a special type of database table which contains the results set of an SQL query. It is a critical database concept DBAs depend upon to maintain high performance for complex SQL queries. So, why

does the design of database security need to pay attention to MQT? Suppose that the DBA creates an MQT M1 based on an SQL query affecting two tables T1 and T2. Further, suppose that table T1 is protected through a set of row permissions and column masks. If such row permissions and column masks are applied during the creation of MQT M1, the content of that MQT becomes dependent on what its creator can or cannot see in base table T1. This would negatively affect the accuracy of the database system's answers. For example, if the database system decides to use M1 to answer a query from a user U1, that user may get more data or less data than what they are authorized depending on whether they have access to more data or less data in base table T1 than the creator of MQT M1. A better approach is therefore to not enforce the row permissions and column masks on T1 during the creation of MQT M1 (or subsequent automatic refresh of its content). But we need to make sure that security is not compromised when doing so. In this context, we have extended the database system such that:

- Upon the creation of an MQT, the database system automatically generates and applies a default row permission with the false predicate "1 = 0". This ensures that direct SQL access to the MQT is blocked (i.e., "1 = 0" always evaluates to false). If certain users have a business need to access the MQT directly, the administrator can create the appropriate row permissions on the MQT to give them access. Any such row permissions or column masks are enforced only during direct access to the MQT.
- When the database system decides to answer a user query from an MQT, it always ensures that any row permissions and column masks on any base table upon which the MQT is defined are automatically carried over and applied on the MQT itself. This ensures that users do not inadvertently get access to data in the base tables for which they are not authorized.

The following experiment illustrates how direct access to an MQT is automatically blocked when its underlying base table is protected by a row permission. This experiment can be consistently repeated on any recent IBM DB2 system. First, we create a table T1 with 2 integer columns A and B. We then insert 3 rows into this table, namely (1,1), (2,2) and (3,3). Next, we create an MQT M1 based on table T1. When we run the statement SELECT A FROM M1, we get the exact same data in base table T1. On the other hand, if we protect T1 with a row

permission and retry that exact same statement, we now get zero rows returned. This is because our design automatically protects the MQT M1 to guard against data leakage. Below are the exact steps.

```
create table T1 (A int, B int);
insert into T1 values (1,1), (2,2), (3,3);
create table M1 (a, b) as (select A, avg(B) from T1 group by A)
        data initially deferred refresh deferred maintained by system;
refresh table M1;
select A from M1;
create permission P1 on T1
       for rows where A = 1
       enforced for all access
      enable;
select A from M1;
```

3.5.3 Database Triggers

A database trigger is a critical database concept which applications depend upon to preserve data integrity. For example, a banking application may decide to use a trigger to ensure that each time a client's balance is updated in the clients table, a row is inserted into the statements table to record that particular withdrawal or deposit transaction. So, why does the design of database security need to pay attention to database triggers? Consider the banking application example above. Suppose that the clients table is protected with a set of row permissions and column masks. If such row permissions and column masks are blindly applied, then it may not be possible to update the statements table as the required input data could have been filtered out or masked. Clearly, this approach would negatively impact data integrity.

A better approach is therefore to not enforce the row permissions or column masks on the clients table. However, not doing so may affect security as the data in the clients table now becomes visible to any triggers defined on such table and may be abused. In this context, we have extended the database system by introducing the notion of a *secure trigger*. By default, a database trigger is not secure, but the administrator can alter the trigger's definition to mark it secure. This means that the administrator vouches for the trigger as trusted and can be applied on a table protected with row permission or column mask constructs. Secure triggers enable database security and triggers to coexist in harmony.

3.6 Performance Evaluation

We have conducted 4 different assessments during our performance evaluation. The assessments were conducted using IBM DB2, extended with our fine-grained authorization model, deployed on a dedicated AIX system with 8 processors @ 1452 GHz and 32GB of RAM. This is a fully dedicated system (CPU, memory, networking and storage) running only our experiment to ensure performance data stability. The time elapsed for a given query is measured from the time the query is submitted to the time the results are returned. Before a query is run, the database system is activated to ensure a fresh database set up. The query is run several times. The first run is discarded from the statistics as the database bufferpool (i.e., database cache) is cold.

- **Assessment 1**: The goal of this assessment is to measure the impact to performance when an application chooses to delegate fine-grained authorization to the database. One of the key advantages of our fine-grained authorization model is that it relieves applications from the burden of enforcing fine-grained authorization by delegating such task to the database. But it is important that this reduction in application complexity does not result in any significant performance drawbacks for the application. This assessment confirmed that applications can safely delegate the enforcement of fine-grained database authorization to the database with no performance concerns.
- **Assessment 2**: The objective of this assessment is to measure the scalability of column masks. Linear scalability has been confirmed by this assessment.
- **Assessment 3**: The goal of this assessment is to verify the independence of column masks. This assessment has shown that the impact of all column masks defined on a table is never higher than the sum of the impact of each column mask defined individually.
- **Assessment 4**: The objective of this assessment is to measure the impact of row permissions. This test confirmed that the impact of row permissions is minimum.

3.6.1 Delegating Fine-Grained Authorization to the Database System

Methodology

We have selected TPC-H (Thanopoulou et al., 2012) as the application with which to conduct our assessment. TPC-H is an industry standard benchmark for measuring database performance. It consists of 22 queries representative of decision support systems that examine large volumes of data. The performance metric reported by TPC-H is called the TPC-H Composite Query-per-Hour Performance Metric (QphH) and reflects multiple aspects of the capability of the database system to process queries.

We focused on two scenarios in our assessment. In the first scenario, we created a set of column masks and row permissions on the TPC-H database schema to specify a fine-grained authorization policy. Then, we ran the TPC-H benchmark and measured the QphH. In the second scenario, we created no column masks or row permissions in the database. Instead, we modified the SQL queries, so the same fine-grained authorization is enforced by the application.

Table 3.1 summarizes our findings. The ratio column represents the QphH of the fine-grained authorization policy delegated to the database divided by the QphH when that policy is enforced by the application itself and is plotted in Figure 3.3. The numbers on the x-axis of this figure represent the 22 TPC-H queries referred to in Table 1. That is, 1 represents query Q1, 2 represents query Q2 and so on.

Discussion

Figure 3.3 shows that almost all the TPC-H queries perform the same or better when the policy is enforced by the database than by the application. More specifically, 13 queries performed fairly the same in both scenarios. 8 queries performed better when the fine-grained authorization policy is enforced by the database system (i.e., the ones where the ratio column is coloured in green in Table 3.1). The improvement observed ranges from 8 to 68%. Lastly, for query Q19, we observed a performance degradation of 15% when the fine-grained authorization policy is enforced by the database.

Table 3.1 – Application vs Database Enforcement for TPC-H Queries

TPC-H Query	QphH Application Enforcement (a)	QphH Database Enforcement (b)	Ratio (b/a)
Q1	1158.8	370	0.3193
Q2	19.7	12	0.6091
Q3	2350.6	2321.6	0.9877
Q4	6105.6	6103.4	0.9996
Q5	7352.6	6371.5	0.8666
Q6	27.8	25.6	0.9209
Q7	16654.1	16657.5	1.0002
Q8	884.2	882.5	0.9981
Q9	9653.8	9475.7	0.9816
Q10	8376.5	8367.3	0.9989
Q11	138.7	127.5	0.9193
Q12	112.6	113.6	1.0089
Q13	103.5	105.7	1.0213
Q14	22.8	14.4	0.6316
Q15	26.7	18.3	0.6854
Q16	24.3	24	0.9877
Q17	336.3	336.2	0.9997
Q18	288.5	291.9	1.0118
Q19	93.6	107.6	1.1496
Q20	73.9	70.8	0.9581
Q21	9655.1	9644.6	0.9989
Q22	90.9	32.9	0.3619

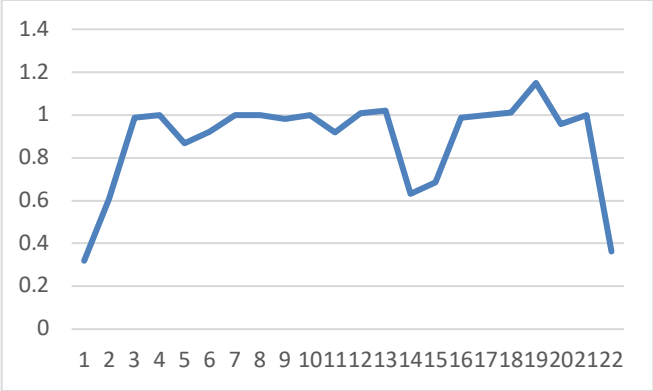


Figure 3.3– Ratio of database vs application enforcement for TPC-H queries

There are two main reasons for the results observed. First, the order in which predicates are evaluated is important, particularly for table joins. For example, consider the following query where tables T1 and T2 are joined on column C1: “SELECT * FROM T1 INNER JOIN T2 on T1.C1 = T2.C1”. When a row permission is enforced by an application, the application will modify the query above by adding the row permission predicates to the SQL text directly as follows: “SELECT * FROM T1 INNER JOIN T2 on T1.C1 = T2.C1 AND <row permission predicate>”. Recall from section 3 that we extended the SQL compiler so that, by default, the row permissions predicates are evaluated first on the table to guard against potential data leakage by any unsafe predicates in the query. So, when the database enforces the fine-grained authorization policy, the query would actually look as follows within the SQL compiler “SELECT * FROM (SELECT * FROM T1 WHERE <row permission predicate>) INNER JOIN T1 on T1.C1 = T2.C1”. However, when there are no unsafe predicates in the query, we do not restrict the SQL compiler optimizer component from moving the row permission predicates higher or lower in the query graph if it leads to a better query execution plan. This was the case in our testing as we had no unsafe predicates. The only situation where the SQL compiler optimizer component did not move the predicate was for query Q19. This is because the row permission defined on the table did not refer to any data in the table itself as it was a simple rule to check whether or not the user issuing the query were a member of a given role. Consequently, the optimizer selected a merge-join instead of a hash-join (Bruno et al., 2014) (Balkesen et al., 2013). Normally, the merge-join would have performed better but because the row permission did not actually filter any rows, the merge-join ended up being more expensive, thus the observed degradation in query Q19.

The second reason for the results observed is how column masks are processed. When the database system enforces a column mask, it does so internally within the actual query graph built by the SQL compiler. So, when the same column appears multiple times within a query the SQL compiler does not need to duplicate the column masks. However, when the fine-grained authorization policy is enforced by the application, the rules representing the column mask end up being duplicated in the SQL query text as the application can only work with SQL. This explains the performance gain observed when the fine-grained authorization policy is enforced by the database.

Our tests have shown that enforcing the fine-grained database authorization policy by the database has not resulted in any significant performance drawbacks for the application. This means that the gains in security and the reduction in application complexity do not come at the expense of application SQL workload performance.

3.6.2 Scalability of Column Masks

Methodology

We have created a table T1 with 10 columns, all of the same type. We have populated the table with random data. No indices of any type were created on this table. We have run a "SELECT * FROM T1" as our baseline. Then, we created a column mask on the first column, ran the same query above and measured its performance. We have repeated this process for each of the remaining columns. The column mask created is exactly the same for each column. We have run the experiment twice: One where T1 contains one million rows and another one where it contains ten million rows. Table 3.2 summarizes our findings.

Table 3.2 – Time Elapsed (in seconds)

Test	1,000,000 rows	10,000,000 rows
Baseline (No Masks)	4.58	44.26
1 Mask	4.73	45.97
2 Masks	4.74	46.45
3 Masks	4.83	46.85
4 Masks	4.82	47.06
5 Masks	4.87	47.48
6 Masks	5	48.28
7 Masks	4.97	48.8
8 Masks	5.02	49.01
9 Masks	5.08	49.96
10 Masks	5.10	50

Discussion

Figure 3.4 shows that for both the one million and ten million rows cases, the execution time of our query scales almost in a linear manner as the number of

masks increases. This confirms our expectation as our design and implementation of column masks did not introduce any additional logic for coordinating the execution of multiple masks when they are present on a given table. Essentially, the overhead introduced is only the one associated with the execution of the actual rule expressed in the column mask definition itself. In our experimentation, the rule was checking user membership in a role to decide whether they see the actual column value or a masked version thereof. It used the built-in SQL function VERIFY_ROLE_FOR_USER. This function is highly optimized. It keeps an in-memory list of users to roles mappings, making it very fast to decide whether or not a user is a member in a given role. We introduced this function to support the adoption of our row permissions and column masks as security best practices advocate for simplifying the management of authorization by assigning privileges to roles and assigning users to roles. Authorization then simply becomes checking user membership in roles.

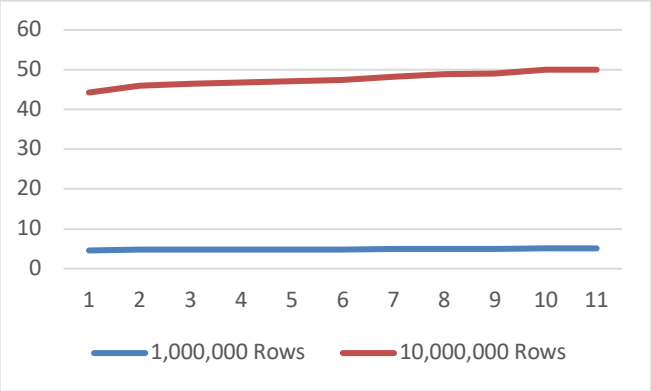


Figure 3.4– Scalability of Column Masks

3.6.3 Independence of Column Masks

Methodology

We have created three column masks on the CUSTOMER table in the TPC-H database schema: A simple column mask, an intermediate column mask, and complex column mask. The simple column mask is similar to the column mask shown in Example 2. It makes use of a single call to function VERIFY_ROLE_FOR_USER to check whether the user is a member of the given role. The intermediate column mask has four calls to the VERIFY_ROLE_FOR_USER function. Lastly, the complex column mask is similar to the intermediate one but has a sub-select statement on top of that.

Our base line is a “SELECT * FROM CUSTOMER” query with no column masks defined on the CUSTOMER table. We ran this query, measured the elapsed time, and then performed the following tests:

- Run the same query with only the simple column mask enabled.
- Run the same query with only the intermediate column mask enabled.
- Run the same query with only the complex column mask enabled.
- Run the same query with all three column masks enabled.

Table 3.3 shows the time elapsed for each test when the CUSTOMER table contains one million rows, and ten million rows respectively. Table 3.4 shows the difference compared to the baseline for each of the tests conducted.

Table 3.3 – Time Elapsed (in seconds)

Test	1,000,000 rows	10,000,000 rows
Baseline (No Masks)	37.464	371.791
Simple Mask	38.812	387.457
Intermediate Mask	40.356	404.619
Complex Mask	58.592	556.439
All Masks	61.855	589.25

Table 3.4 – Difference with the Baseline

Test	1,000,000 rows	10,000,000 rows
Simple Mask	1.348	15.666
Intermediate Mask	2.892	32.828
Complex Mask	21.128	184.648
Sum of all Masks	25.368	233.142
All Masks	24.391	217.459

Discussion

Figure 3.5 contrasts the sum of the differences to the baseline for each of the simple, intermediate, and complex mask tests with the difference to the baseline for the test where all masks are enabled at the same time for both the one million rows and ten million rows cases. For both cases, we can observe that the difference with the baseline when all masks are enabled at the same time is never higher than the sum of the differences to the baseline for each individual mask.

This confirms our expectation as our column masks design and implementation did not require introducing any coordination when multiple masks are enabled at the same time. The masks are in fact totally independent from each other.

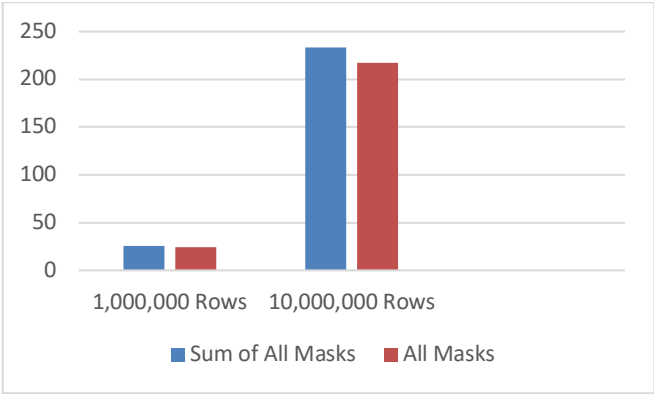


Figure 3.5– Independence of Column Masks

3.6.4 Row Permissions Impact

Methodology

We have created three row permissions on the CUSTOMER table in the TPC-H database schema: One row permission that returns zero rows, one permission that returns 50% of the rows, and another row permission that returns all rows. We have run "SELECT * FROM CUSTOMERS" as our baseline. Then, we run the same query with each of the row permissions above enabled individually (i.e. one row permission at a time). Table 3.5 shows the time elapsed for each test when the CUSTOMER table contains one million rows and ten million rows respectively.

Table 3.5 – Time Elapsed (in seconds)

Test	1,000,000 rows	10,000,000 rows
Baseline (No Permissions)	38.163	380.118
Permission (0 rows)	0.11	3.173
Permission (50% rows)	19.679	169.154
Permission (All rows)	38.679	383.93

Discussion

Figure 3.6 and Figure 3.7 contrast the performance for each of the 3 tests with our baseline for the one million rows and ten million rows respectively. The results

are similar for each case and show that the overhead of row permissions is very minimal. For instance, when the row permission returns all rows, the performance is almost identical to the baseline. This is expected as the rule expressed in the row permission is internally implemented as a predicate. In our case, the predicate includes the built-in VERIFY_ROLE_FOR_USER SQL function. If a DBA decides to deploy their own UDF for use in a row permission definition, the performance implications may be different depending on several factors such as how optimized that UDF is and whether or not it is declared as trusted.

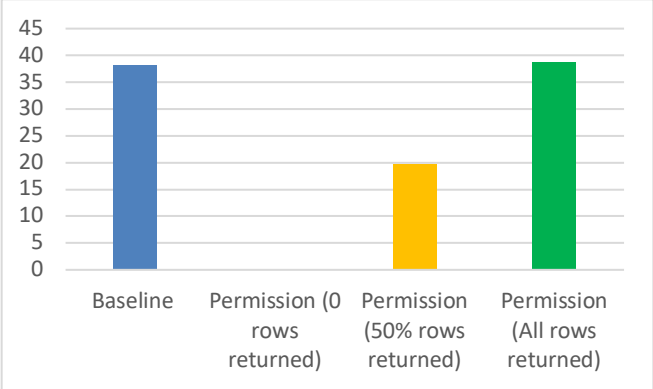


Figure 3.6– Row Permissions Impact (1,000,000 rows)

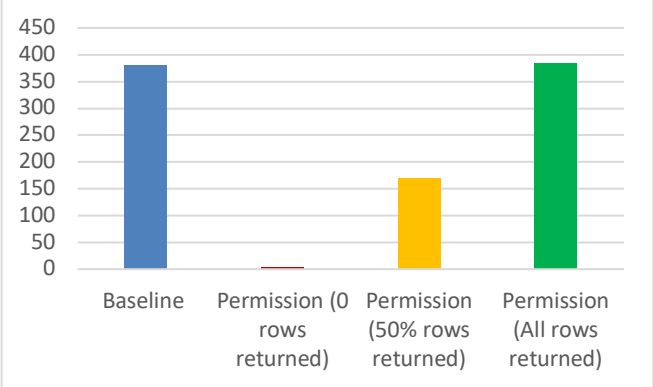


Figure 3.7– Row Permissions Impact (10,000,000 rows)

3.7 Use Case Scenario

We describe how our row permissions and column masks can be applied to meet the needs of a banking application. All the SQL statements and outputs below have been fully verified with our implementation on IBM DB2. These requirements can be summarized as follows:

- Customer service representatives and telemarketers can see all data.
- Tellers can see only the data for their own branch customers.

- The customer account number is accessible only by customer service representatives. All other users can only see the last 4 digits.

Customer information is stored in a table called CUSTOMER and bank employee information is stored in a table called EMPLOYEE_INFO. The SQL statements for creating these two tables are given below.

```
create table customer (account varchar (9),
                       name varchar (20),
                       income int,
                       branch char (1) );
create table employee_info (branch char (1),
                            emp_id varchar (10) );
```

We assume that tables CUSTOMER and EMPLOYEE_INFO are already populated. Their content is given by tables 3.6 and 3.7 respectively.

Table 3.6 – CUSTOMER Table

ACCOUNT	NAME	INCOME	BRANCH
1234-5678	Alice	22,000	A
2345-6754	Bob	71,000	B
3456-1298	Carl	123,000	B
4672-8901	David	172,000	C

Table 3.7 – EMPLOYEE_INFO Table

EMP_ID	BRANCH
Amy	A
Pat	B
Haytham	C

Tellers, customer service representatives, and telemarketers are members of database roles TELLER, CSR, and TELEMARKETER respectively. SELECT privilege to the CUSTOMER table is granted to these three roles. Users Amy, Pat and Haytham are a teller, a customer service representative and a telemarketer respectively. The SQL statements for setting up these roles are given below.

```
create role teller;
grant select on customer to role teller;
grant role teller to user amy;
create role csr;
grant select on customer to role csr;
grant role csr to user pat;
create role telemarketer;
grant select on customer to role telemarketer;
grant role telemarketer to user haytham;
```

To implement the first rule which states that customer service representatives and telemarketers can see all customers, the following row permission must be created.

```
create permission csr_row_access on customer
        for rows where verify_role_for_user (USER, 'csr') = 1 or
                              verify_role_for_user (USER, 'telemarketer') = 1
        enforced for all access
        enable;
```

To implement the second rule which states that tellers can only see customers of their own branch, the following row permissions must be created. The sub-select in the permission definition ensures that the customer's branch and the teller's branch match.

```
create permission teller_row_access on customer
        for rows where verify_role_for_user (USER, 'teller') = 1 and
                              branch = (select branch from employee_info
                                    where emp_id = USER)
        enforced for all access
        enable;
```

To implement the third rule, the following column mask is created. The mask ensures that when the user is not a member of the CSR role, they see only the last 4 digits of the account number. The rest of the digits are replaced by "X"s for them (masked out).

```
create mask csr_column_access on customer
        for column account
        return case when verify_role_for_user (USER, 'csr') = 1
                        then account
                        else 'XXXX-' || SUBSTR(ACCOUNT,5,4)
                    end
        enable;
```

Now that the row permissions and column masks have been defined, any future access to the CUSTOMER table will see the database system automatically enforce the security policy. Table 3.8 contrasts the output when the application issues the query “SELECT * FROM CUSTOMER” for users Amy, Haytham and Pat respectively.

When the application issues that query on behalf of user Amy, the database only returns the rows for customers from branch A, which is where Amy works. Note that the account number is masked out because Amy is not a member of the CSR role.

On the other hand, when the application issues the exact same query on behalf of user Haytham, the database returns all the rows in the table which is in accordance with the first rule because Haytham is a telemarketer. Note that the account number is still masked out because Haytham is not a member of the CSR role.

Lastly, when the same query is issued on behalf of user Pat, all the rows in the table are returned and the account number is not masked out because Pat is a member of the CSR role.

Table 3.8 – Outputs for Users Amy, Haytham and Pat

USER	ACCOUNT	NAME	INCOME	BRANCH
Amy	XXXX-5678	Alice	22,000	A
Haytham	XXXX-5678	Alice	22,000	A
	XXXX-6754	Bob	71,000	B
	XXXX-1298	Carl	123,000	B
	XXXX-8901	David	172,000	C
Pat	1234-5678	Alice	22,000	A
	2345-6754	Bob	71,000	B
	3456-1298	Carl	123,000	B
	4672-8901	David	172,000	C

This example has shown how the application logic can remain very simple. In all 3 user situations, the application simply issues the simple “SELECT * FROM CUSTOMERS” SQL query. The database system automatically applies the fine-grained authorization rules, relieving the application from this burden, which in turn contributes to reducing the complexity of the application.

3.8 Conclusion

We have introduced a fine-grained database authorization model which allows applications to safely delegate the burden of fine-grained authorization to the database system, where it is enforced more effectively. In particular, we have shown how the trusted context mechanism introduced allows applications to propagate user identities to the database system in a controlled manner in multitiered environments, strengthening overall database security. We have also shown how the trusted context mechanism can be used to provide control on when the application privileges can be exercised which helps protect against potential abuse of the application user ID (application bypass).

The row permission, column mask, and trusted context concepts introduced also enable organizations to implement zero-trust security for database systems. Row permissions and column masks allow such organizations to ensure that data is accessed based on “*need-to-know*”, which is a key tenet of zero-trust security. Additionally, trusted contexts help organizations implement the “*verify and never trust*” zero-trust security tenet, by having the database system verify additional attributes before granting access to a user or application.

In our future work, we plan to focus on facilitating the adoption of our fine-grained database authorization model. For example, defining a column mask is a very easy task once you know which column to define it on. But in some situations, this knowledge may not be available (e.g., a database inherited through a merger or an acquisition). This is where data classification would be useful. The main challenges in this context would be to investigate how to do the data classification on the database efficiently and accurately. Additionally, we want to explore machine learning for automatically generating the appropriate row permissions and column masks. Machine learning has been explored for detecting threats (Alloghani et al., 2020), (Aljawarneh et al., 2018), (Aldwairi et al., 2012), but here we would like to explore it for fine-grained authorization policy recommendation.

Chapter 4: Enhanced Data Encryption

This chapter provides a summary of the research portfolio in database encryption. It is based primarily on a research publication “Holistic Database Encryption” that is given in Appendix B. The chapter first reviews traditional data encryption methods in database systems and contrasts them with holistic database encryption. Then, it introduces holistic database encryption and discuss its implementation in a commercial database system.

4.1 Introduction

Data encryption is a powerful control for protecting sensitive data. For database systems, traditional data encryption approaches come in many shapes and forms, but each with a set of challenges forcing organizations to make compromises either on the security side or on the database side when adopting such approaches. Clearly, a better approach was needed so that data encryption coexists in harmony with fundamental database tenets such as performance and compression, thus eliminating the need for organizations to make any such compromises. In this thesis, this better approach is referred to as "***holistic database encryption***". Section 4.2 reviews the traditional approaches and contrasts them with holistic database encryption. Section 4.3 introduces holistic database encryption while Section 4.4 discusses its implementation in IBM DB2. Lastly, Section 4.5 concludes this chapter.

4.2 Related Work

Traditional database encryption solutions can be divided into four main categories: Column encryption, tablespace encryption, file system encryption and self-encrypting disks.

Column encryption allows an application to encrypt data at the column level in a database table (Benfield et al., 2001). Typically, the database system provides a set of built-in UDFs to give applications the tools to encrypt and decrypt data stored in database table columns. The main advantage of column encryption is security as the column data remains encrypted from the point of entry in the application all the way down to the storage and vice versa. However, this gain in security comes at a cost. For example, because standard encryption is not order preserving, queries with range predicates cannot benefit from index-based access plans to limit the data to read from the table. Instead, the database system is forced to read the entire table to evaluate the query. Additionally, encrypting data in the application limits the value of database compression as compression will be left to operate on encrypted data which typically does not include patterns. Last but not least, column encryption complicates adoption of the solution for pre-packaged applications where the organization does not own the source code for the application. Holistic database encryption improves over column encryption. It does not interfere at all with query execution and therefore does not negatively impact the performance of range queries. It takes place within the database

kernel itself, after compression has occurred, thus allowing organization to benefit from both database encryption and compression. Also, holistic database encryption is totally transparent to applications. Lastly, while holistic database encryption does not protect data in transmission, this can be easily mitigated by ensuring TLS is turned on to secure the channel between the database system and applications.

Tablespace encryption provides the DBA with the option to indicate that data in given tablespace must be automatically encrypted by the database system itself (Boobal, 2018). It improves over column encryption in the same way holistic database encryption does. However, tablespace encryption can leave data vulnerable to attacks. For example, DBAs often create materialized query tables (MQT) to speed up the execution of data warehousing queries (Zilio et al., 2004). In doing so, data from an encrypted tablespace may find itself in another tablespace which the DBA omitted to specify it must be encrypted upon creation. This would leave the data in the MQT vulnerable to attacks. Additionally, data in the system-defined tablespaces is not encrypted. For example, the system catalogues typically include statistics information which the database system relies upon to generate optimal access plans for executing queries. Some of these statistics include actual data values such as the most frequent values in a column, and the highest and lowest values in that column. This would unfortunately leave such data vulnerable to attacks. Holistic database encryption improves over tablespace encryption because it automatically encrypts the database as a whole including any system-defined or temporary tablespaces. It simplifies database administration for the DBA and avoids the risk of creating vulnerabilities when inadvertently moving data to an unencrypted tablespace, such as when creating an MQT for boosting query performance purposes.

File system encryption is an indirect mechanism to encrypt the database objects by encrypting their underlying physical files. Examples of file system encryption solutions include the Encrypted File System (EFS) on the IBM AIX systems (IBM, 2018), the EFS on the Microsoft Windows systems (Microsoft, 2018), and eCryptfs on Linux systems (Halcrow, 2007). File system encryption can also be provided using add-on tools such as Vormetric Transparent Data Encryption (Vormetric, 2018) and Gemalto Protect File (Gemalto, 2018). These tools deploy an agent on the operating system where data encryption is needed.

The agent is a kernel module which extends that operating system to enable file encryption. Compared to the native file system encryption above, these tools allow an organization to manage file system encryption uniformly across a heterogeneous operating systems environment. File system encryption also improves over column encryption and shares the same benefits as tablespace encryption. However, it is not supported on all file systems. For example, the IBM EFS solution for AIX systems is only available on JFS2 file systems. This limits the set of database deployments which can benefit from this solution. Additionally, some database deployments choose to write their data directly to raw devices bypassing the file system altogether. In this case, a file system encryption solution cannot be used to encrypt the database objects. Also, native file system encryption provides no protection against privileged users on the operating system. As long as the file permissions allow access, such users can easily browse the content of the encrypted files. Lastly, the division of responsibilities between the DBA who is responsible for database administration and the SA who is responsible for system administration may introduce security vulnerabilities. For example, if a DBA creates a new tablespace and places it on an unencrypted file system, the content of that tablespace will not be encrypted and would be left open to attacks. Holistic database encryption addresses the file system encryption challenges above. It is built in the database kernel itself, so it is available anywhere the database is deployed. Also, being part of the database kernel means that database can choose to write to raw devices directly and still ensure data is encrypted. Additionally, the database content is not vulnerable to users browsing the file systems since that content can only be decrypted by the database itself. Last but not least, the division of responsibilities between the DBA and SA does not create any vulnerabilities as the database is automatically and fully encrypted by the database kernel itself.

Self-Encrypting Disks (SED) is another indirect mechanism for encrypting the database objects by relying on the circuitry built into the hard drive itself to encrypt the data (Dufrasne et al., 2016). Examples of SED include IBM DS8000, Seagate SED, and Hitachi SED. They too improve over column encryption, share the same benefits with tablespace, file system and holistic database encryption. They provide the broadest coverage as the full disk is encrypted. With respect to performance, SED actually add no overhead to the CPU as the encryption is done by the hard drive itself. SED do have certain drawbacks, however. Firstly, they

are a disruptive and expensive approach as an organization would need to purchase these new devices and replace existing hard drives. Secondly, SED provide no protection against privileged users on the operating system. As long as the file permissions allow access, such users can easily browse the content of the encrypted files. Holistic database encryption addresses these challenges because it is part of the database itself, so it creates no disruptions to the organization's IT infrastructure. It also ensures that the content of the database is not vulnerable to users browsing the file system as that content can only be decrypted by the database itself.

4.3 Holistic Database Encryption

The first objective of holistic database encryption is to ensure that the full database content is automatically encrypted by the database system itself. This needs to include not only user-defined tablespaces, system-defined tablespaces and temporary tablespaces, but also data in transaction logs and database backups. This is to ensure that no vulnerability is inadvertently introduced as with the traditional methods discussed earlier. The second objective is to ensure that the first objective is met without negatively impacting core database tenets such as application transparency, schema transparency, performance, compression or availability. It is achieving both of these objectives together that allows organizations to adopt the solution without having to make any compromise either on the security side or the database side, as with the traditional methods. In order to achieve these two objectives, three requirements need to be carefully considered: Encryption run-time placement, encryption run-time processing and encryption key management. The design considerations for each of these requirements is discussed below.

4.3.1 Encryption Run-Time Placement

Holistic database encryption places the run-time processing of encryption right above the I/O layer inside the database kernel. This ensures that all data is encrypted regardless of whether it is tablespace data, transaction logs data or backups data. Additionally, injecting encryption processing this deep inside the kernel renders encryption totally transparent to database schemas and applications. Recall that the lack of transparency to applications and database schemas was one of the major drawbacks of column encryption. Also, being right above the I/O layer ensures that encryption does not interfere at all with query

execution so the SQL query compiler does not need to impose any restrictions on itself when selecting an efficient execution plan for a query as it does in the case of queries with range predicates when using column encryption. Last but not least, this encryption run-time placement means that encryption comes after compression which is the right order to ensure that organizations benefit from both database compression and encryption. Recall that with column encryption, the order is reversed which limits the value of database compression.

4.3.2 Encryption Run-Time Processing

Encryption run-time processing refers to the encryption and decryption functions. Data is encrypted when it is pushed out to storage and decrypted when it is retrieved from storage. These two functions take place right above the I/O layer as indicated above. Encryption and decryption require carefully considering three choices: The choice of the encryption algorithm, the choice of the encryption algorithm key size and the choice of the encryption granularity.

Since database encryption is bulk encryption, symmetric algorithms are the natural choice. While holistic database encryption can support any block cipher, AES (Chandra et al., 2014) was selected as the default block cipher as it is the standard algorithm. Holistic database encryption chose 256 bits as the default key size to ensure that the solution is safe against potential future attacks by quantum computers (Shor, 1997). In fact, a quantum computer running Grover's algorithm (Grover, 1996) renders AES 256 bits security equivalent to AES 128 bits so choosing 128 bits AES Keys would not be a good practice as that reduces to 64 bits security. AES supports several modes. Holistic database encryption uses Cipher Block Chaining (CBC) as that is more secure than Electronic Code Book (ECB) for example. However, CBC requires maintaining an Initialization Vector (IV) and this affects the choice the encryption granularity. Holistic database encryption uses the "*page*" as the encryption granularity. A page is a 32KB of data containing the rows of a database table. A database table may consist of several such pages. The choice of the page as the encryption granularity versus the data row is evident as calling the encryption function for each row in the page would result in a higher performance overhead. The choice of a "*chunk*" of pages would in theory have been better. In fact, the database kernel I/O layer writes a chunk of pages at a time for performance reasons. However, given the chaining nature of CBC, this meant that if the database kernel

needs to decrypt page 4, it will first need to decrypt pages 1, 2 and 3. Thus, the page level granularity was chosen for performance reasons.

4.3.3 Encryption Key Management

Figure 4.1 shows the architecture of holistic database encryption as implemented in IBM DB2. It uses two levels of keys: A Data Encryption Key (DEK) and a Master Key (MK). The DEK is the key used to actually encrypt the database content and is fully managed within the database system. For tablespace data, the DEK is stored together with the rest of the database configuration. However, this would not be sufficient for transaction logs as these are typically needed when the database is offline and need to be recovered. Consequently, transaction logs have their own DEK that is stored within the transaction logs themselves. This is also true for database backups which may need to be restored by a totally different database instance.

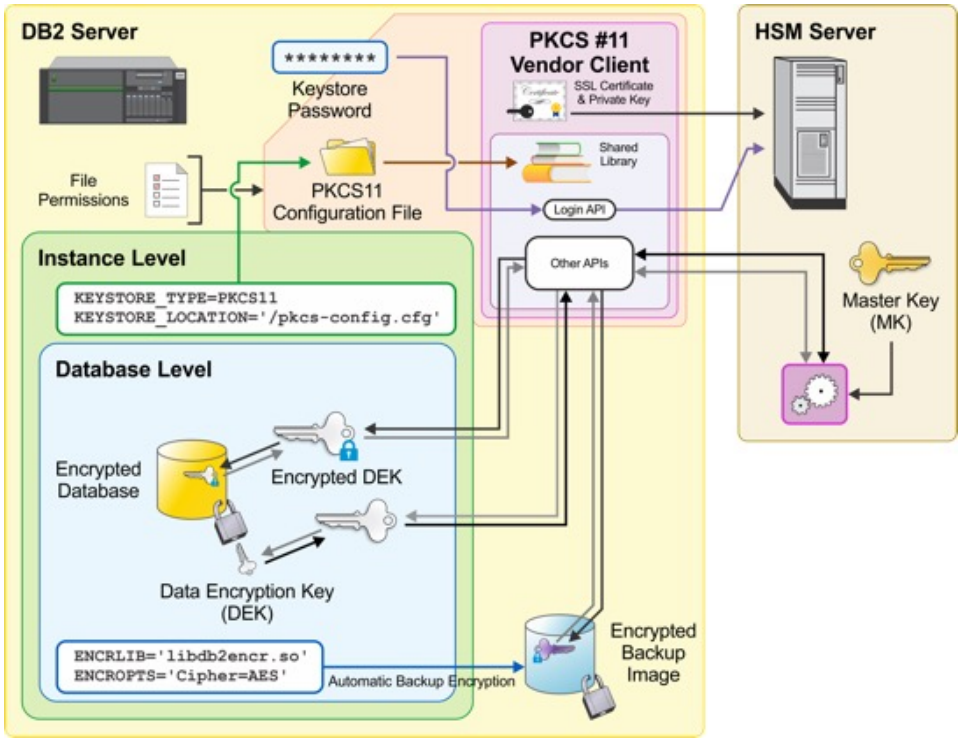


Figure 4.1– Holistic Database Encryption Architecture

The MK is a Key Encrypting Key (KEK). It is used to protect the DEK and is stored externally such as in a Hardware Security Module (HSM). There are three reasons for this choice: Security, performance and availability. Storing the MK externally ensures a better security in case the database system itself is compromised (i.e., the attacker will not have access to both the data and the MK). The two levels of keys also ensure availability and performance in key rotation

scenarios. In fact, rotating the MK is straightforward as it only means decrypting the DEK with the old MK and re-encrypting it with the new MK. On the other hand, a single level of keys would have meant re-encrypting the entire database with the new key. This would affect availability if it is done offline or performance if it is done while the database is online serving the needs of its applications.

4.4 Implementation

This section discusses the implementation of holistic database encryption in IBM DB2 and shows the actual interfaces for adopting the solution.

4.4.1 Enabling Encryption for a Database

The DB2 CREATE DATABASE command was extended so that DBAs can choose to enable encryption when creating a new database. For example, the DB2 command below creates a database called test which will be encrypted using AES as the encryption algorithm, 256 bits as the encryption key size, and a master key whose unique identification is db-mk.

```
create database test
encrypt cipher AES key length 256
master key label db-mk
```

When processing the command above, the database system internally generates a random 256 bits DEK and call out to the key management system that has been set up for this database to encrypt the DEK before safely storing it within the database configuration structures. Later on, when the database is started to serve applications, the database system internally calls out to the key management system to decrypt the DEK with the MK, and then uses the DEK for transparently encrypting and decrypting data as it is written to and read from the storage system.

4.4.2 Rotating the Database Master Key

Like user passwords, encryption keys need to be changed periodically in order to minimize the risks when a key is compromised. This process is called key rotation. The key rotation frequency is dictated by compliance requirements, corporate requirements or both.

Holistic database encryption extended the IBM DB2 interfaces by introducing a new stored procedure which DBAs can use to rotate the database MK as

required. For example, the DB2 stored procedure call below instructs the database system to rotate the MK.

```
CALL admin_rotate_master_key ('new-db-mk');
```

When processing the stored procedure above, the database system performs the following actions:

- Decrypt the DEK with the current MK.
- Re-encrypt the DEK with the new MK as identified by it is unique label new-db-mk.
- Update the database configuration structures to reflect the changes above.

If a new MK unique identifier has not been provided when calling the stored procedure above, the database system will automatically generate a new MK and assigns a unique identifier to it.

4.4.3 Taking an Encrypted Database Backup

Holistic database encryption extended the DB2 BACKUP DATABASE command so that DBAs can choose to enable encryption when backing up a database. Encryption for the backup is actually automatically enabled when the underlying database is encrypted. But the explicit option in the BACKUP DATABASE command itself gives DBAs the option to still encrypt a backup even when the underlying database is not encrypted. For example, the DB2 command below encrypts a backup for a database called test2 using AES as the encryption algorithm, 256 bits as the encryption key size, and a master key whose unique identification is db-mk2.

```
backup database test2
encrypt encrlib db2backupencrlib
encropts 'Cipher=AES:Key LENGTH=256:Master Key Label=db-mk2'
```

When processing the command above, the database system internally generates a random 256 bits DEK, uses that DEK to encrypt the payload piece of the backup, call out to the key management system to encrypt the DEK with the MK identified with unique identifier db-mk2, and safely store the encrypted DEK and related meta-data in the header piece of the backup.

4.4.4 Performance Considerations

The performance evaluation shows that two factors affect the impact of holistic database encryption. The first factor is the availability of hardware acceleration in the CPUs where the database system is deployed. Holistic database encryption automatically detects and exploits a number of hardware acceleration for cryptographic operations built into modern CPUs such as the Intel Advanced Encryption Standard New Instructions (AES-NI) and the IBM Power8 in-core support for AES. The second factor is how insulated the database workload from an increase in the latency of physical I/O requests. Database workloads can be insulated for this purpose through standard database tuning. For example, a DBA can increase the buffer pool size so that database queries do not have to wait on physical I/O. Enabling page prefetchers is another tuning option a DBA can perform to avoid having queries wait on physical I/O. Following standard database tuning, the encryption overhead observed is typically in the single digits for data warehouse workloads on systems with exploitable hardware acceleration for cryptographic operations. It is therefore recommended that enterprises deploy the solution on systems where such hardware acceleration for cryptographic operations is available.

4.5 Conclusion

This chapter has provided a summary of the research portfolio in database encryption. It has reviewed the traditional approaches for database encryption and showed how the holistic database encryption proposed improves over such approaches. Holistic database encryption has been implemented in IBM DB2 and is relied upon by several organizations from across the world to protect their sensitive data without having to make compromises either on the security side or the database side as the solution coexists in harmony with fundamental database tenets as described in this chapter. The research portfolio publication "*Holistic Database Encryption*" is given in Appendix B.

Chapter 5: Enhanced Mandatory Access Control

This chapter provides a summary of the research portfolio in mandatory access control for database systems. It is a synthesis of the research publications given in Appendix C. The chapter introduces a new multi-purpose implementation of mandatory access control for database systems which improves over traditional implementations. This new solution is not limited to the pure Multilevel Security (MLS) semantics as the traditional approaches and can be used more broadly. The chapter also shows how the multi-purpose implementation of mandatory access control introduced can be used to enforce fine-grained authorization to XML documents such as those stored in XML columns in database tables.

5.1 Introduction

For database tables, Mandatory Access Control (MAC) can be thought of as a special form of fine-grained authorization where each row is tagged with a security label representing its classification (e.g., TOP SECRET), each user is assigned a security label representing their authorization (e.g., SECERT) and the access rules are the standard Multilevel Security (MLS) "*No Read Up*" and "*No Write Down*" rules. While this model is suitable for the US intelligence and defense use cases, it remains a very rigid implementation that is rarely applicable elsewhere. Clearly, a better approach is needed to broaden the applicability of MAC implementations in database systems. This better approach is referred to as "***A Multi-Purpose MAC Implementation for Database Systems***". It extends the traditional MAC implementations for database systems with the required flexibility in order to broaden its applicability. Section 5.2 reviews traditional MAC implementations and contrasts them with the multi-purpose MAC implementation. Section 5.3 introduces and discusses the multi-purpose MAC implementation in IBM DB2. Section 5.4 shows how the multi-purpose MAC implementation can be used for fine-grained authorization in XML documents. Lastly, Section 5.5 concludes this chapter.

5.2 Related Work

Traditional MAC implementations for database systems have focused on MLS. The MLS model was originally introduced by Bell and LaPadula (Rjaibi et al., 2004). It is defined in terms of objects and subjects. For database tables, an object is a row in that table and a subject is a user requesting access to such row. Both objects and subjects are assigned a security label representing their classifications and authorizations respectively. A security label consists of two components (Rjaibi et al., 2004): A hierarchical component, usually referred to as *level* and a non-hierarchical component, usually referred to as *compartments*. The level specifies the sensitivity of the data. For example, a military organization might define the following levels: Top Secret, Secret and Confidential. Compartments are used to categorize the data. For example, a military organization might define the following compartments: Navy, Army and Marines.

A security label L1 is said to dominate a security label L2 if and only if the following two conditions are true:

1. The level component of L1 is greater than or equal to that of L2.

2. The compartments component of L1 includes the compartments component of L2.

For all data access, the MLS model enforces the following two rules:

1. No Read Up: A subject is allowed a read access to an object if and only if the subject's security label dominates the object's security label.
2. No Write Down: A subject is allowed a write access to an object if and only if the object's security label dominates the subject's security label.

The most noticeable implementations of MLS in database systems include: Trusted Oracle (Oracle, 1992), Oracle Label Security (which replaced Trusted Oracle) (Oracle, 2019), Informix OnLine/Secure (Informix, 1993) and IBM DB2 for z/OS MLS (Rayns et al., 2007). The research portfolio publications "***An Introduction to Multilevel Secure Database Systems"*** and "***A Multi-Purpose Implementation of Mandatory Access Control in Database Systems***" (Appendix C) cover the traditional MLS implementation in more details. But the common theme across all these traditional implementations is that they are all a rigid implementation and are rarely applicable in scenarios where the pure MLS semantics is not desired. The multi-purpose MAC implementation introduced in this thesis is a flexible implementation that is not limited to pure MLS semantics and can be used more broadly as illustrated in Section 5.4.

5.3 A Multi-Purpose MAC Implementation for Database Systems

The first objective of the multi-purpose MAC implementation is to give DBAs the tools to define the types of security labels and access rules that best suit their needs as opposed to forcing the pure MLS semantics on them. This objective is achieved through the SQL extensions discussed in section 5.3.1. The second objective is to ensure that access to labelled data is enforced transparently, securely and in accordance with the access rules specified. This is achieved through the extensions made to the SQL Compiler discussed in section 5.3.2. Lastly, the third objective is to enable the database system to integrate with an external MAC system, if so desired, to centralize security labels and access rules management. This is analogous to integrating with an LDAP server for user authentication. This objective is achieved through the SQL and SQL Compiler extensions discussed in section 5.3.3.

5.3.1 SQL Extensions

The multi-purpose MAC implementation extended SQL with the following new concepts to give DBAs the tools to specify security label types and access rules:

- **Security Label Component**: This is the building block for security labels. It is essentially a set of elements which can be either ordered or un-ordered. In an ordered set, the order in which the elements appear is important. The rank of the first element is higher than that of the second element, and so on.
- **Security Label Type**: As a table schema defines the set of columns for rows in that table, a security label type defines the set of security label components that make up a security label. For example, the classical MLS security label can be obtained by creating a security label type that consists of two security label components, one that is ordered representing the level component and one that is un-ordered representing the compartments. On the other hand, if a DBAs wishes to use security labels as data tags, they can simply create a security label type that consists of a single un-ordered security label component, where each element represents the desired tag.
- **Security Label Access Policy**: This is where the access rules are defined. The access rules bring together an *access label* and a *row label*. An access label is a security label that is granted to a database user. A row label is a label that is assigned to a row in a table. The general form of an access rule is "*access label component-name <operator> row label component-name*". The <operator> varies depending on whether the component-name is an ordered set or an un-ordered set. For ordered set, it can be anyone of the relational {=, <=, <, >, >=, !=}. For un-ordered sets, it can be anyone of the set operators {INCLUDE, INTERSECT}.
- **Exception**: In some situations, a user may need to be granted an exception from a certain access rule in a security label access policy. For example, to allow the user to do a bulk load of data in a table.
- **Labelled Table**: A labelled table is a table that is associated with a security label policy. Such table will be automatically augmented with a new column to hold the security label for each row. The security label is internally transformed into a binary representation for efficient comparisons during access enforcement.

Example

The following example shows how a DBA can specify security label and access rules definitions to match the pure MLS semantics.

```
// create security label components
create security label component LEVEL
using ordered set {'TOP SECRET', 'SECRET', 'CLASSIFIED'};
create security label component COMPARTMENTS
using unordered set {'MARINES', 'ARMY', 'NATO'};

// create security label type
create security label type MLS
components LEVEL, COMPARTMENTS;

// create security label access policy
create security label access policy MLS-POLICY
security label type MLS
read access rule rule 1
        access label level >= row label level
read access rule rule 2
        access label compartments INCLUDE row label compartments
write access rule rule 1
        row label level >= access label level
write access rule rule 2
        row label compartments INCLUDE access label compartments;

// create a labelled table
create table T1 (A INT, B INT)
security label access policy MLS-POLICY;
```

5.3.2 Access Enforcement

The multi-purpose MAC implementation extended the SQL compiler to ensure that when a labelled table is accessed, the access rules specified in the security label access policy are observed. The extensions made are similar to the ones done for row permissions and column masks (Figure 3.2 in Chapter 3) with a couple of additional considerations. The first consideration is with respect to how the user access label and potential exceptions are acquired. If these are acquired at query compilation time, it will affect the caching of query execution plans. Some database systems such as IBM DB2 cache the execution plan for an SQL query

so that the next time it is submitted it does not need to go through the SQL compilation process again and performance is better. Imagine that the security label and exceptions are acquired during compilation time, this means that logic will need to be added to always check that the security label and exceptions of the user submitting the query are the same as the ones recorded during query compilation time. Otherwise, this could result in a security issue such as a user getting more data than what their security label and exceptions permit. For this reason, the execution plan generated by the SQL Compiler includes a new logic to always acquire security labels and exceptions during run-time. This ensures security as well as performance as the cached query execution plan can be reused without having to worry about any potential differences between the credentials of the user ID under which the query was compiled and the user ID that is running the query.

The second consideration is index-only query execution plans. To access a database table, the SQL compiler typically chooses between three options: (1) Accessing the table directly and fetch the desired rows, (2) accessing an index to first identify the IDs of the rows that need to be fetched and then access only the pages containing such rows, or (3) accessing only an index on the table. The latter is possible when all the desired columns are part of the index key. For large tables, this is usually an advantageous option as indexes are usually smaller than the table on which they are defined. For labelled tables, the row label is required to decide whether or not the user should be given access to the row. Therefore, to ensure that index only plans still work, we extended the database system so that each time an index is created, the row label column is automatically included in the index key.

5.3.3 Enterprise integration

Some tools such as IBM's Resource Access Control Facility (RACF) (Winnard et al., 2015) provide an MLS implementation where user security labels can be centrally managed much like an LDAP server allows user authentication to be managed centrally. RACF can also be used for access decisions. That is, given a row label and a user ID, it can return true or false indicating whether or not the given user can have access to that row according to the MLS rules. To enable integration with such enterprise solutions, the multi-purpose MAC was further extended as follows:

- The labelled table SQL syntax was extended to allow DBAs to indicate that the security label access policy is managed by an external system and provide the connection details to such system so it can be called for access decision responses during access enforcement.
- The SQL compiler was extended to recognize this special case and inject logic to query the external system for an access decision, passing on the row label and the ID of the user attempting the access.

To minimize the overhead of calling out to the external system, the multi-purpose MAC implementation introduced an access-decision cache. This cache records the responses from the external system for each pair of user ID and row label. The SQL compiler was then modified so that it consults this cache before making a call to the external system. Table 5.1 illustrates the access-decision cache. For example, user Amy is allowed read access to rows labelled with security label L1, but not user Pat.

Table 5.1 – Access-Decision Cache

USER ID	ROW LABEL	TABLE	ACCESS TYPE	RESPONSE
Amy	L1	T1	READ	YES
Pat	L1	T1	READ	NO
Haytham	L2	T1	WRITE	YES

5.4 Applying Multi-Purpose MAC for XML Fine-Grained Authorization

While the row permission, column mask and security label concepts introduced so far in this thesis permit enforcing fine-grained authorization at the database table row and column levels, they do not extend to enforcing fine-grained authorization for XML documents stored within a column in such database tables. For example, suppose the XML document in Figure 5.1 is stored in some database table column. A user who is authorized to access that column will see the entire XML document as opposed to the subset of such document they are authorized to see.

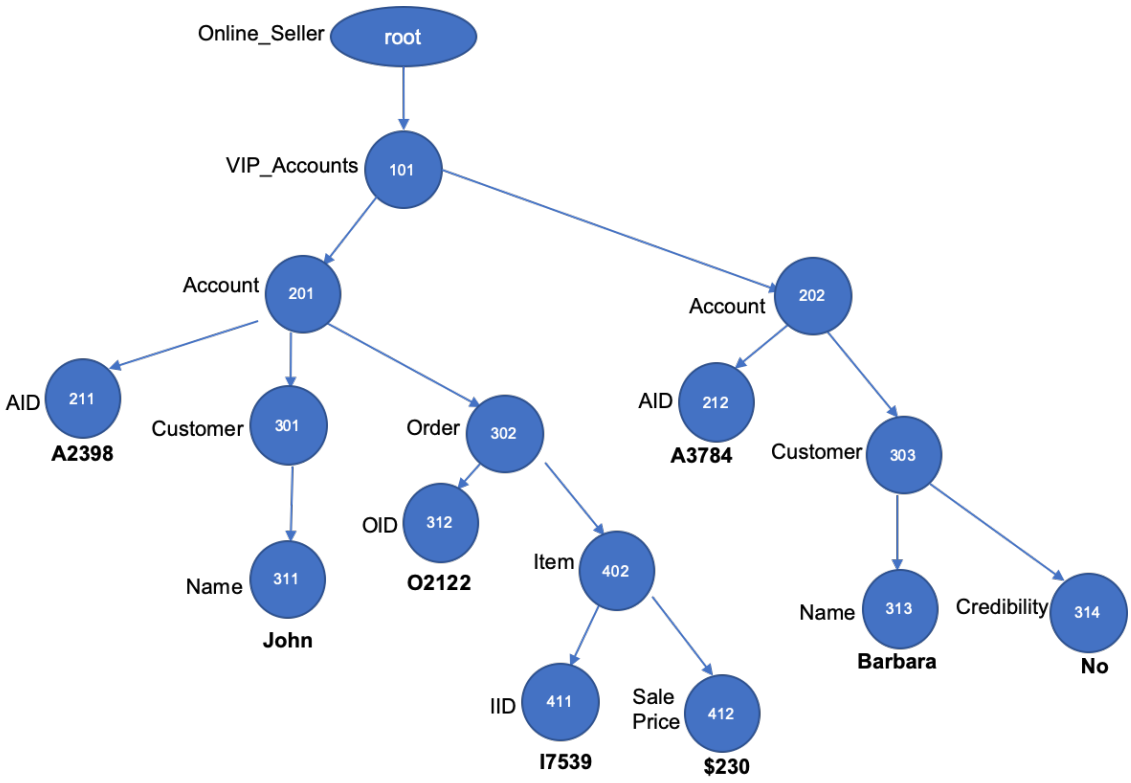


Figure 5.1– Example XML Document

XML fine-grained authorization can be divided into two main categories. The first category considers an XML node as the smallest unit of protection (Bertino et al., 2001) (Bertino et al., 2002) (Bhatti et al., 2004). The approaches in this category differs in terms of how privileges are propagated. Some methods block access to the entire subtree rooted at a forbidden node while others would allow access to nodes in the subtree but mask out the forbidden ancestor node. The second category considers the ancestor-descendent and sibling relationships between XML nodes as the smallest unit of protection (Zhang et al., 2006). For example, blocking access to the account node "202" in Figure 1 can be achieved by blocking the relationship to its ancestor node, the relationship to its descendant nodes and the relationship to its sibling nodes. In both categories, the smallest unit of protection (i.e., a node or a relationship) is specified through an XPath expression (Clark et al., 2006). The Multi-Purpose MAC solution introduced earlier could be used with either node-based or relationship-based fine-grained authorization approaches. The research portfolio publication "***Inter-Node Relationship Labelling: A Fine-Grained XML Access Control Implementation Using Generic Security Labels***" shows that relationship-based approaches improve over node-based approaches from a security perspective. For example, the subtree rooted at node "101" represents VIP

account types. Therefore, knowing that an account (e.g., “201”) belongs to that subtree reveals that this is a VIP account. Suppose that the relationship between nodes “101” and “202” needs to be protected. With node-based approaches, access to node “201” would reveal that node “202” is a VIP account since it is a sibling to node “201”. This issue can be addressed using a relationship-based approach by protecting the ancestor-descendant relationship between nodes “101” and “202” as well as the sibling relationship between nodes “201” and “202” while allowing the ancestor-descendant relationship between nodes “101” and “201”. The rest of this section discusses the methodology for using the Multi-Purpose MAC solution to enforce fine-grained authorization to XML documents using the relationship-based approach.

5.4.1 Methodology

The methodology is analogous to how security labels are used to control access to rows in a labelled table. In the same way a row label protects a row in a labelled table, a *path label* protects a specific path in an XML document. The path label consists of a single security label component which can take anyone of the following three values:

- **Existence:** Attaching a path label with this value to a relationship between two nodes permits users to know that such two nodes are related but does not reveal any other details. For example, suppose an existence path label is attached to the relationship between the account with AID A2398 and its customer name in Figure 5.1. A query that wants to return all the accounts’ AIDs that have a customer name would return AID A2398 but will not reveal that the customer name is “John”. An example of such query in XPath is: *//Account[Customer/Name]/AID*.
- **Value:** Attaching a path label with this value to a relationship between two nodes permits users to know that such two nodes are related including the actual details of such nodes. For example, suppose a value path label is attached to the relationship between the account with AID A2398 and its customer name in Figure 5.1. An XPath query such as *//Account[AID=”A2398”]/Customer/Name* would reveal that “John” is the customer associated with that account. Evidently, if a relationship is not accessible under an existence path label, then it is not accessible under a value path label either.

- **Null:** Attaching a path label with value to a relationship between two nodes means that this relationship is fully accessible. This is the default.

The proposed ATTACH SQL statement allows DBAs to attach a path label to the desired relationships in an XML document. This can be either an ancestor-descendant relationship or a sibling relationship. For example, the following SQL statement attaches an existence path label to the relationships between account nodes (i.e., ancestor) and their customer name nodes (i.e., descendants).

```
attach existence ancs //Account desc /Customer/Name
```

The following example shows how to associate a security label access policy with a table T1 which includes a column B of type XML.

```
// create security label components
 create security label component level
 using ordered set {'EXISTENCE', 'VALUE', 'NULL'};

 // create security label type
 create security label type XML
 components level;

 // create security label access policy
 create security label access policy XML-POLICY
 security label type XML
 read access rule rule 1
       access label level >= path label level
 write access rule rule 1
       access label level = path label level;

 // create a labelled table
 create table T1 (A INT, B XML)
 security label access policy XML-POLICY;
```

5.4.2 Access Enforcement

As discussed in section 5.3.2, access enforcement is implemented within the SQL compiler. To handle queries on XML document, the following additional considerations are observed by the SQL compiler. First, the XPath query

semantics is internally changed as follows to take into account any path labels attached (Zhang et al., 2006).

1. If a child axis occurs, the evaluation follows a parent-child path.
2. If a descendant-or-self axis occurs, the evaluation follows an ancestor-descendant path.
3. If a preceding-sibling axis occurs, the evaluation follows a preceding-sibling path.
4. If a following-sibling axis occurs, the evaluation follows a following-sibling path.

As with access to labelled tables, the user access labels are acquired at runtime (as opposed to compilation time) to ensure that queries on XML documents also benefit from caching of query execution plans. The general access enforcement algorithm can be summarized as follows.

1. Fetch the user access labels and exceptions from the system catalogues.
2. For all paths accessed
    a. If it is a read access and the read access rules do not allow the access, skip the path.
    b. If it is a write access and the write access rules do not allow the access, skip the path.

5.5 Conclusion

This chapter has provided a summary of the research portfolio in mandatory access control for database systems. It has introduced a new multi-purpose implementation of mandatory access control for database systems which improves over traditional implementations. This new solution is not limited to the pure MLS semantics as the traditional approaches and can be used more broadly. The chapter has also shown how the multi-purpose implementation of mandatory access control can be used to enforce fine-grained authorization to XML documents such as those stored in XML columns in database tables. The multi-purpose mandatory access control solution has been implemented in both IBM DB2 and Informix.

Chapter 6: Towards Zero-Trust Database Security

Zero-trust security is an information security framework which states that organizations should not trust any entity inside or outside of their perimeter at any time. This chapter explores both the direct and indirect means through which the same data in a database system can be accessed and the challenges they pose to adhering to the basic tenets of zero-trust security. It is based primarily on the research publications “Towards Zero-Trust Database Security Part 1” and “Towards Zero-Trust Database Security Part 2” which were fully developed during the PhD registration period (Appendix B). The chapter then shows how the concepts introduced earlier in this thesis such as row permissions, column masks, trusted contexts and holistic database encryption come together to equip database systems with the controls necessary to enable enterprises to effectively implement zero-trust security for their database installations.

6.1 Introduction

Gartner estimates that the worldwide spending on Cybersecurity in 2018 was around 114 billion US dollars, which represents an increase of 12.4% compared to 2017 (Gartner, 2019). Unfortunately, despite this significant spending, data breaches continue to occur and are becoming more and more costly. For example, the Ponemon Institute's 2018 Cost of a Data Breach Study found that the global average cost of a data breach was 3.86 million US dollars, an increase of 6.4% compared to 2017 (The Ponemon Institute, 2019). The study also found that the average number of records lost or stolen following a data breach grew 2.2% from 2017. As a result of these alarming statistics, organizations are now turning to zero-trust security to better protect their assets and reduce risk.

Zero-trust security is an information security framework which states that organizations should not trust any entity inside or outside of their perimeter at any time (Gilman et al., 2017). It assumes that untrusted entities exist both outside and inside the enterprise network. The main tenets of zero-trust security can be summarized as follows:

1. Tenet 1: Ensure all requests to access resources are always verified, regardless of where they originated from.
2. Tenet 2: Grant access to resources based on "need-to-know" and strictly enforce access control.
3. Tenet 3: Monitor and audit all user activities.

While zero-trust security for networks and identity management systems has received a great deal of attention (Gilman et al., 2017), (Centrify, 2019), very little focus has been devoted to zero-trust security for database systems (Rjaibi et al., 2019). This is concerning as database systems contain an enterprise's most critical data and are often the primary subject of attacks by both internal and external threats. The rest of this chapter is organized as follows. Section 6.2 introduces the database threat model and explores both the direct and indirect means through which the same data in a database can be accessed. Next, Section 6.3 shows how the concepts introduced earlier in this thesis equip database systems with the tools necessary to effectively address the challenges posed by the direct and indirect means through which data can be accessed. Section 6.4 explores the notion of separation of duties as another critical

foundation to fully enable zero-trust database security. Lastly, Section 6.5 shows a concrete example to illustrate how the row permission, column mask and trusted context concepts introduced in this thesis come together to meet an enterprise's zero-trust database security requirements.

6.2 Database Threat Model

This threat model focuses on the direct and indirect means for accessing data in a database and the challenges they pose to adhering to the basic tenets of zero-trust security discussed above. The model assumes that enterprises are adhering to basic database security hygiene such as user authentication, auditing and TLS, which are standard features on all major database systems. The model also assumes that standard operational policies such as operating system and database system software vulnerability patching are in place.

The same data in a database can be accessed in two different ways: Directly or indirectly. Direct access occurs using standard database interfaces such as Structured Query Language (SQL). This can be divided into two scenarios:

1. Interactive database access: This access is typically performed by database administrators using an interactive interface offered by the database system such as SQL. It is usually used to carry out administrative tasks such as granting database privileges.
2. Application database access: This is the most common database access scenario. It involves end users interacting with an application which in turn interacts with the database system to execute requests on behalf of those end users.

The key issue with interactive database access is ***privilege abuse***. For example, a DBA might abuse their privileges to access sensitive employee data such as salary and bonus information. The application database access poses two key issues. The first one is ***application bypass*** where, for example, the application administrator abuses the application's database credentials to make changes to the database that are contrary to the application's business logic. The second issue is the ***loss of user identity*** which diminishes the value of database auditing to demonstrate compliance and to hold users accountable for their actions. This issue stems from the fact that applications use a generic user ID to access the database on behalf of all users as opposed to the actual user identity.

Indirect access takes place when a user bypasses the database system altogether. This is the most dangerous type of access as it completely bypasses all database access control and auditing. This can be divided into two scenarios:

1. ***File system access***: This access occurs when a user chooses to access the data directly on the file system using operating system commands.
2. ***Storage media access***: This access occurs when a user recovers the data from the actual storage media such as a stolen or lost hard drive.

Table 6.1 summarizes the challenges direct and indirect access to data pose to adhering to the basic tenets of zero-trust security. Figure 6.1 summarizes the database threat model.

Table 6.1 – Zero-trust database security challenges

Issue	**Type of data access**	**Zero-trust security tenet affected**
Privilege abuse	Direct access	Tenets 1 and 2
Application bypass	Direct access	Tenets 1 and 2
Loss of user identity	Direct access	Tenet 3
File system access	Indirect access	Tenets 1 and 2
Storage media access	Indirect access	Tenets 1 and 2

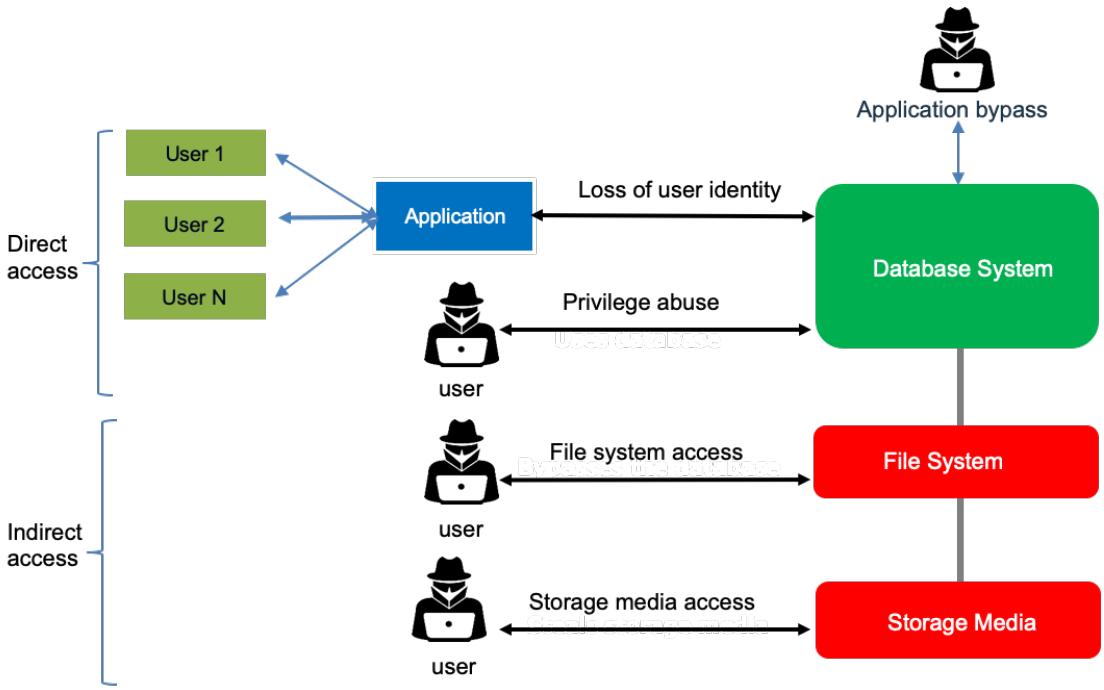


Figure 6.1– Database threat model.

6.3 Addressing Direct Data Access Challenges

As discussed in section 6.2, privilege abuse, application bypass and the loss of user identity are the key challenges to adhering to the basic tenets of zero-trust security when it comes to the direct data access use case. This section describes how the concepts introduced in this thesis address these challenges.

6.3.1 Privilege Abuse

Historically, database systems have been designed such that the DBA had access to all data in all tables in the database. Clearly, this model does not prevent privilege abuse. Intuitively, fine-grained database authorization can be thought of as the ideal solution for preventing privilege abuse as it controls access at the row, column or cell level, thus ensuring that users have access to only the subset of the data for which they are authorized. However, fine-grained database authorization comes in many forms and not all such forms adequately protect against privilege abuse.

Chapter 3 introduced row permissions and column masks. It also showed how these concepts improve over traditional database views and application-based security. In particular, row permissions and column masks are data-centric and cannot be bypassed like database views and application-based security. Similarly, Chapter 5 introduced the multi-purpose MAC implementation and showed how it improved over traditional MLS implementations by providing more flexibility around the specification of security label types and access policies. Both row permissions and column masks, and the multi-purpose MAC implementation effectively address the privilege abuse challenge. Figure 6.2 contrasts all the fine-grained authorization options. Row permissions and column masks are ranked slightly higher than the multi-purpose MAC because it is more flexible. In fact, their authorization rules are expressed in SQL, thus provide more flexibility than rules that only manipulate security labels. Therefore, row permissions and column masks are most suitable for addressing the privilege abuse challenge.

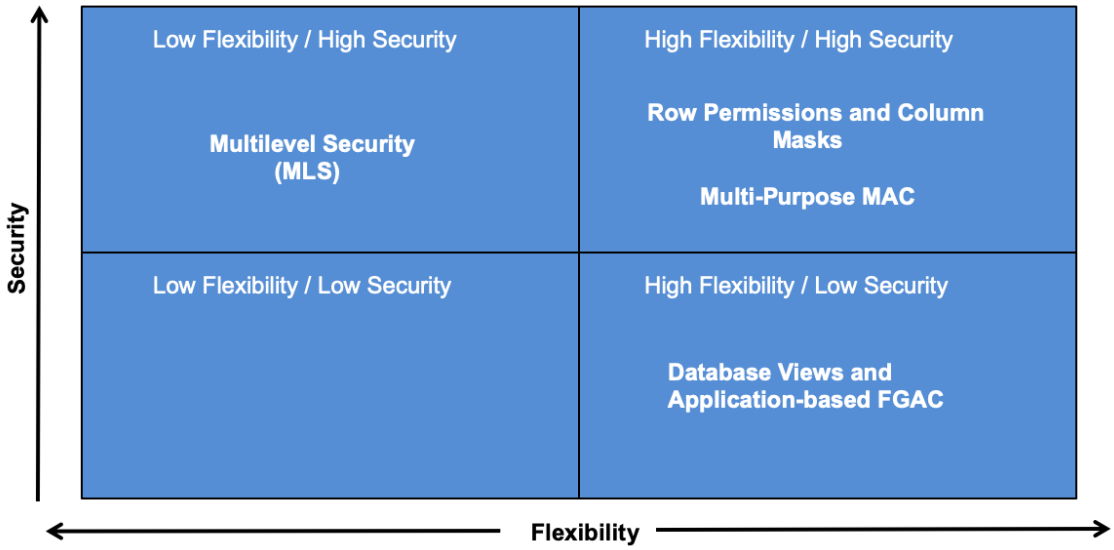


Figure 6.2– Fine-grained database authorization.

6.3.2 Application Bypass

A key drawback to application-based security is that applications can be bypassed. For example, a malicious application administrator can choose to abuse the application's database credentials in order to access the database directly, thus bypassing the application altogether. The malicious application administrator can then gain access to sensitive data or make modifications to the database that are contrary to the application's business logic.

This application bypass is made possible because traditional database authorization does not provide control around when a particular privilege can be exercised. Section 3.4 in Chapter 3 introduced the concept of trusted context. One of the benefits of this new concept is the ability to address application bypass by linking database privileges to a trusted context. When a privilege is linked to a trusted context, a user can exercise that privilege only when they are interacting with the database system within the confines of a trust relationship. Application bypass can be addressed in this manner by requiring the database system to authorize the application's user ID only and only if additional attributes have been verified such as the IP address of the application server and the application's digital certificate. Therefore, an application administrator who wishes to abuse the application credentials by connecting to the database outside the scope of the application will find it much harder to do so.

6.3.3 Loss of User Identity

In multitiered database environments, the application interacts with the database system using a generic user ID identifying the application itself, and not the actual end users. One major implication of this is diminished user accountability. Typically, database users are held accountable for their actions through auditing. Unfortunately, when the application uses a generic user ID for all database accesses, the database audit log will only show that user ID with no references to the actual end user behind the application.

The trusted context concept introduced in Chapter 3 is a formal mechanism for defining a trust relationship between the database system and an external application based on a series of attributes such as the application's user ID, the IP address of the application server and the application's digital certificate. One of the capabilities that an application gains once it is working within the confines of that trust relationship is the ability to switch the current user on a given database connection. This enables the application to propagate the user identity to the database where it is used for access control and auditing purposes, and thus addressing the loss of user identity problem. The high-level steps for leveraging the trusted context concept to address the loss of user identity problem can be summarized as follows:

1. The database security administrator creates a trusted context object to define a trust relationship between the application and the database.
2. The application establishes a trusted connection with the database.
3. Before issuing any request to the database on behalf of an end user, the application switches the current user of the connection to the new user. This automatically propagates the end user identity to the database where it is used for all access control and auditing till the application switches user again.

6.4 Addressing Indirect Data Access Challenges

As discussed in section 6.2, file system access and storage media access are the key challenges to adhering to the basic tenets of zero-trust security when it comes to the indirect data access use case. A powerful countermeasure to protect against indirect access is data encryption as encrypted data is of no value to an attacker. However, data encryption for database systems comes in many forms and not all such forms of encryption effectively protect against indirect data

access. There are also performance implications that need to be taken into account when selecting an encryption solution for a database.

Chapter 4 contrasted the traditional database encryption methods with the new solution proposed in this thesis: Holistic database encryption. Figure 6.3 provides a different perspective for contrasting these approaches. SEDs and file system encryption provide the broadest coverage (i.e., they encrypt entire disks or file systems), but they only protect against storage media access. In other words, these methods do not stop a user from browsing the database files using operating system commands. Tablespace encryption, holistic database encryption and column encryption protect against both storage media access and file system access. When these methods are employed, a user browsing the database files using operating system commands will only see encrypted data, which is of no value to them. Column encryption, however, is intrusive to application and negatively affects performance. Tablespace encryption may create a vulnerability when a DBA inadvertently moves data from an encrypted tablespace to an unencrypted one, or when data is held in temporary tablespaces. Therefore, holistic database encryption is most suitable for protecting against indirect access and in turn adhering to the basic tenets of zero-trust security.

Table 6.2 summarizes the issues around the direct data access and indirect data access use cases. It also shows how the concepts introduced in this thesis solve these issues. Row permissions and column masks address the privilege abuse issue. Trusted contexts and in particular the conditional authorization aspect of it solve the application bypass issue. The user identity propagation aspect of trusted contexts solves the loss of user identity issue in multitiered database environments. Finally, holistic database encryption addresses the issues around file system access and storage media access.

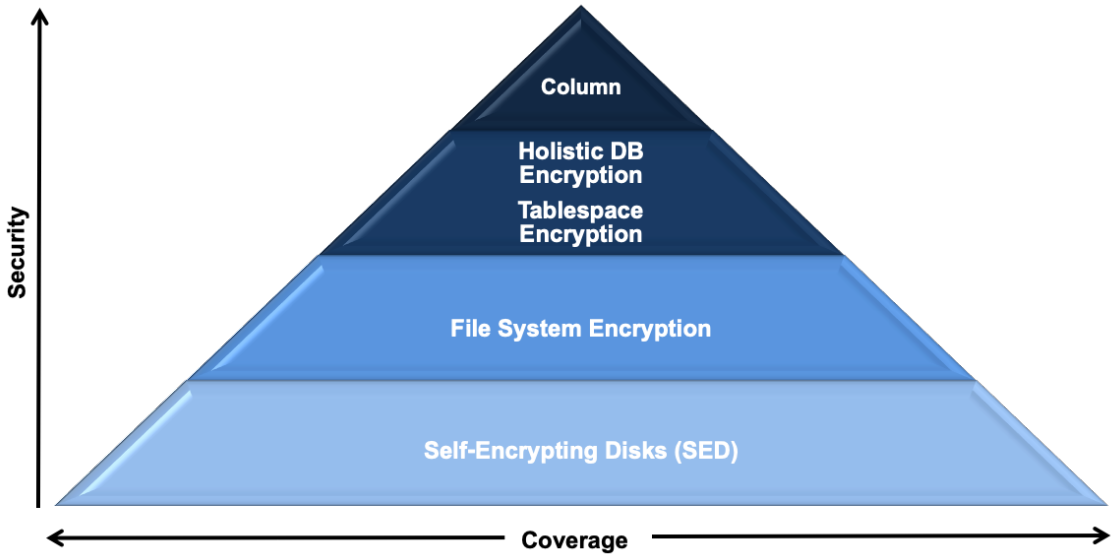


Figure 6.3– Database encryption.

Table 6.2 – Zero-trust database security challenges and solutions.

Issue	Type of data access	Zero-trust security tenet affected	Solution
Privilege abuse	Direct access	Tenets 1 and 2	- Row permissions - Column masks
Application bypass	Direct access	Tenets 1 and 2	Trusted contexts - Conditional authorization
Loss of user identity	Direct access	Tenet 3	Trusted contexts - User identity propagation
File system access	Indirect access	Tenets 1 and 2	Holistic database encryption
Storage media access	Indirect access	Tenets 1 and 2	Holistic database encryption

The performance evaluations for row permissions/column masks and holistic database encryption have been covered in Chapter 3 and Chapter 4 respectively. For trusted contexts, the performance evaluation (Bruni et al., 2007) has shown that the overhead is in the low single digits regardless of whether or not user authentication is required during the switch user processing. Intuitively, this is expected as the database system reuses the existing database connection as opposed to creating a new one to process requests on behalf of a new user.

6.5 Separation of Duties

Historically, database systems have been designed such that DBAs manage all aspects of the database including security and auditing. Additionally, DBAs have

had implicit access to all data in all tables in the database. With the rise of internal threats as a security concern equally important to external threats (Verizon, 2017), this traditional model makes it difficult for organizations to fully adhere to zero-trust database security. It is therefore critical that database systems be extended to provide the capabilities to allow organizations to vest security administration and database administration into two non-overlapping roles so separation of duties can be enforced.

Consequently, during the research, design and implementation of row permissions, column masks, trusted contexts and holistic database encryption, I have also made the following corollary enhancements:

1. Redesigned the role of the DBA to remove the implicit ability to access all data in all tables as well as the ability to manage database security and auditing.
2. Vested the ability to manage database security and auditing into a new and independent database role, called Security Administrator (SECADM) (Chen et al., 2008). In this context, row permissions, column masks, trusted contexts and holistic database encryption are solely managed by the SECDAM.
3. Implemented the new SECADM role and also ensured that such role cannot make any privilege grants to itself either directly or indirectly through membership in a role or a group. This automatically covers row permissions and column masks as they are an additional level of control on top the required table level privileges.

With this enhancement in place, organizations can now vest database security and database administration into two separate roles, enabling them to remove any notion of inherent trust in DBAs and consequently fully adhere to zero-trust security for their database systems. It is paramount that organizations consider separation of duties as they choose the type of database system to adopt because not all database systems necessarily provide the required capabilities to enforce separation of duties.

6.6 Example Scenario

The goal of this example is to show in great detail how the concepts introduced in this thesis enable organizations to implement zero-trust database security. The

example will cover all the enhancements made, namely holistic database encryption, trusted contexts (both the conditional authorization and the user identity propagation aspects), row permissions and column masks. It builds upon the example shown in Section 3.7 of Chapter 3. All SQL statements shown here are the actual interfaces for the contributions made in this thesis as they have been fully implemented in IBM DB2.

The example represents a banking application which stores and manages customer sensitive data. It is a classical 3-tier application. The first tier is the set of bank employees using the application through a standard web browser. The second tier is the application server running the actual application logic. We assume that the application server's IP address is 72.137.255.114. Lastly, the third tier is the database where the application stores and manages customer data. Table 6.3 summarizes the bank's security policy which must be implemented by the application.

Table 6.3 – Banking application security policy.

#	Requirement	Rationale
1	All customer data must be protected against online threats.	Protect against users browsing the database files on the operating system – file system access.
2	All customer data must be protected against offline threats.	Protect against loss or theft of storage media – storage media access.
3	All customer data must be accessed through the application only.	Protect against customer data changes outside the application business logic – Application bypass.
4	All application user activities must be tracked.	Ensure application users are held accountable for their actions – Loss of user identity.
5	All customer data must be accessed on a need-to-know basis.	Protect against DBAs abusing their privileges – Privilege abuse.
6	Customer service representatives and telemarketers can see the data about all customers.	Ensure customer data is accessed on a need-to-know basis.
7	Tellers can see only the data for their own branch customers.	Ensure customer data is accessed on a need-to-know basis.
8	The customer account number is accessible only by customer service representatives. All other users can only see the last 4 digits. The rest of the account number digits are masked out for such users.	Ensure customer data is accessed on a need-to-know basis.

As discussed earlier in this chapter, requirements 1 and 2 (file system access and storage media access) are addressed through holistic database encryption. Below is the actual SQL statement to create the banking application's database:

```
create database AppDB
encrypt cipher AES key length 256
master key label AppDB-MK;
```

The SQL statement above instructs the database system to create a new database called AppDB and ensure that data stored within that database is automatically encrypted using AES with a key that is 256 bits in size. The master key label AppDB-MK is a unique identifier for a key wrapping key that is stored outside the database such as a Hardware Security Module (HSM). This master key is used to protect the Data Encryption Key (DEK) that is stored inside the database. The DEK is the key that is actually used to encrypt and decrypt the data stored in the database. The database system automatically interacts with the HSM each time it needs to encrypt or decrypt the DEK with the master key.

To address requirements 3 and 4, we need to create a trusted context object in the database to define a trust relationship between the database and the application. Below is the actual SQL statement to create such trusted context.

```
create trusted context AppCtx
based upon connection using system authid AppUserID
attributes (address '72.137.255.114'
            encryption 'SSL')
default role DBCONNECT
with use for Amy without authentication,
             Pat without authentication,
             Haytham without authentication
enable;
```

There are two parts to the SQL statement above. The first one is the definition of a trust relationship between the database and an application that is identified by a series of attributes, namely the application's user ID (AppUserID), the IP address from which the application initiates database connections (72.137.255.114) as well as the nature of the protection over the communication channel between the application and the database (SSL). Each time a database

connection is attempted using AppUserID, the database system automatically assesses the additional attributes of that incoming database connection. If the incoming connection attributes fully match the attributes specified in the definition of trusted context AppCtx, then that incoming connection automatically gains two key capabilities that are not available to it otherwise. More specifically:

1. The incoming connection inherits the role DBCONNECT. This is the role that actually authorizes the database connection to take place. In other words, if the application administrator chooses to abuse the application credentials to access the database directly, the database system will not allow that database connection to take place. This is how requirement 3 is addressed.
2. The incoming connection inherits the ability to switch user IDs on the database connection established. In this specific example, the application will be allowed to switch the current user on the established database connection to users Amy, Pat and Haytham. So, each time the application needs to issue database requests on behalf of any of these users, it will first switch the current user on the connection to the desired user ID. This is how requirement 4 is addressed.

Requirements 5, 6, 7 and 8 are about direct data access. This is where row permissions and column masks come into play. Once these are in place, customer data will be accessed based on the bank's application security policy (requirements 6, 7 and 8). Additionally, DBAs cannot abuse their privileges to access such customer data because row permissions and column masks are enforced uniformly across all users regardless of their privilege or authority (requirement 5). The row permissions and column masks SQL to implement requirements 6, 7 and 8 has already been given in Section 3.7 in Chapter 3 and will not be repeated here.

6.7 Conclusion

This chapter explored both the direct and indirect means through which the same data in a database system can be accessed and the challenges they pose to adhering to the basic tenets of zero-trust security. Privilege abuse, application bypass and the loss of user identity in multitiered database environments represent the key challenges for the direct access scenarios while file system access and storage media access represent those for the indirect access use

cases. The chapter then showed how the concepts introduced in this thesis around holistic database encryption, trusted context's conditional authorization, trusted context's user identity propagation, row permissions and column masks come together to equip database systems with the controls necessary to help enterprises effectively implement zero-trust security for their database installations. Lastly, the chapter provided a concrete example showing the actual interfaces for the concepts introduced in this thesis as implemented in a commercial database system.

Chapter 7: Conclusion and Future Work

This chapter summarizes the thesis and outlines potential future research directions in database security. Row permissions, column masks, trusted contexts and holistic database encryption are the key contributions made in this thesis. They equip database systems with the controls necessary to help enterprises effectively implement zero-trust database security. Data classification, machine learning and homomorphic encryption are three potential research directions for database security. Data classification would help facilitate the adoption of concepts such as row permissions and column masks. Machine learning can be used to detect unknown threats such as SQL injections. Homomorphic encryption would remove any privacy concerns when adopting cloud database services.

7.1 Introduction

Database systems are at the core of an organization's information system. They store the organization's most critical assets such as client personal data, patient healthcare records, employee personal data, financial transactions, intellectual property and are consequently the primary target of attacks by both insiders and outsiders. They are also the subject of numerous compliance mandates such as the European General Data Protection Regulation (GDPR), the US Health Insurance Portability and Accountability Act (HIPAA) and the Payment Card Industry Data Security Standard (PCI DSS). These compliance mandates combined with the continuous increase in data breaches and the rise of internal threats as a security concern equally important to external threats (Verizon, 2017) have driven organizations towards zero-trust security to better protect their assets and reduce risk.

7.2 Key Contributions

This thesis enhanced database systems to equip them with the necessary controls to help enterprises effectively implement zero-trust database security. The most noticeable contributions in this regard can be summarized as follows:

1. Holistic database encryption: This solution enables organizations to effectively protect their data including the file system access and storage media access challenges discussed in Chapter 6. Unlike other database encryption methods (Rjaibi, 2018), this solution does not force organizations to make any compromises on either the data side or the security side. For example, unlike column encryption, holistic database encryption does not negatively affect database performance because it does interfere at all with query processing.
2. Row permissions and column masks: This solution enables enterprises to ensure that data is accessed solely on a need-to-know basis. Unlike previous methods (Rjaibi et al., 2020), this solution ensures that the security policy is enforced uniformly across all users regardless of their privilege or authority. This also addresses the privilege abuse challenge discussed in Chapter 6. It also integrates thoughtfully with the rest of the database tenets, so organizations do not have to make any compromises when adopting the solution. For example, the solution harmonically integrates with Materialized Query Tables (MQT) so organizations can still

benefit from the MQTs performance boost without having to compromise database security.

3. Trusted contexts: This solution provides two key benefits. First, it extends database systems with the required controls to address the application bypass and the loss of user identity challenges outlined in Chapter 6. Next, it enables applications to safely delegate the fine-grained authorization policy to the database system where it can be enforced more effectively. Without trusted contexts, fine-grained authorization solutions such as row permissions and column masks are of little value in a multitiered database environments because the database system only sees a generic user ID representing the application itself and not its end users.

Throughout the research, emphasis has been on both innovation and practicality. This is paramount for database systems as security innovations that come at the expense of core database tenets such as performance, integrity, compression or require changes to database applications are unlikely to be adopted by a commercial database system, and even more unlikely to be used in practice by clients. For example, a bank is unlikely to adopt a column masking solution if that requires changing hundreds of applications. Similarly, the bank is unlikely to enable database encryption if that causes a significant performance degradation to a mission critical application or if encryption nullifies the benefits of compression and forces the bank to purchase more storage hardware. In this regard, the enhancements proposed in this thesis have been fully implemented in several commercial database systems such as IBM DB2 and Informix, where they are relied upon by thousands of banking, insurance, retail, government and other types of organizations from around the world to protect their critical data and meet their compliance mandates.

7.3 Future Directions

Database security needs to continue to evolve to facilitate the adoption of security capabilities and address emerging challenges and use cases. In this context, data classification, machine learning and homomorphic encryption are key future directions for database security.

7.3.1 Data Classification

Data classification would facilitate the adoption of fine-grained authorization solutions such as the row permission and column mask concepts introduced in

this thesis. The concepts themselves are very easy to implement once the data to protect is known. But in some cases, the nature of this data may not be known. For example, consider a database inherited from another department or perhaps from an acquisition. The data needs to be analyzed and classified so the sensitive tables and sensitive columns are identified. While data classification tools exist (IBM, 2019), they either take a long time to classify a large database or they are forced to sample the data and create room for false negatives. Building data classification in the database system itself and enable the database to do this automatically and transparently as the data is ingested would help solve this problem. Besides the expected challenges around how to perform data classification in a way that minimizes both false positives and false negatives, it is critical that this data classification does not compromise other key tenets such as database performance.

7.3.2 Machine Learning

Machine learning would enable the database system to address another class of external threats. For example, consider a classical 3-tier application. Suppose that it is an internet facing application and that an external attacker exploits an SQL injection vulnerability in the application. While an SQL injection is an application problem (as opposed to a database problem), the attacker can still compromise the database by fooling the application into executing unintended SQL statements such as retrieving the content of the application's users table or dropping actual database tables. In this case, the database system cannot figure out that it is being attacked since the requests are coming from a legitimate application which holds all the proper privileges to execute the requests it issued. Machine learning can be used to enable the database system to build a model of the database and user activities so that deviations from such model can be detected. For example, if the application suddenly starts downloading massive amounts of data, that may be a sign of an SQL injection attack. Besides the expected challenges around what type of machine or deep learning models are more effective for a database system, the solution must not compromise database performance during model creation or subsequent online updates of such model. While anomaly detection tools based on machine learning exist (Adir et al., 2017), they often lack visibility into full database activities. Therefore, implementing such capability in the database system itself would be more effective as the database system has full visibility into all the user activities it

processes.

7.3.3 Homomorphic Encryption

Homomorphic encryption would allow enterprises to take full advantage of cloud computing. For example, cloud database services relieve enterprises from the burden of deploying, configuring, patching, upgrading, scaling, backing up and recovering database systems. However, despite these significant gains enterprises are still reluctant to adopt these database services. This is due to security concerns around storing sensitive data in the cloud. While virtually all cloud vendors provide encryption solutions for their database services, the mere fact that data is encrypted on the cloud vendor premises means that there is a time at which that sensitive data exists in clear text and may be abused by a malicious entity. The ultimate solution would be to ensure that data is encrypted on premises with keys managed by the client also on premises before it is ingested into the cloud database service. The challenge would then be to enable the database system to perform queries over the data without having to decrypt it first. This is where homomorphic encryption may be able to help. The idea would be to encrypt the data in such a way that the database system can evaluate queries over the encrypted data directly and still return the same results as if the evaluation were done over clear text data. While some research solutions exist (Popa et al., 2011), they tend to restrict the type of SQL that can be executed over the encrypted data. Clearly more research is needed here to ensure that the benefits of homomorphic encryption does not come at the expense of key database tenets such as functionality and performance.

7.4 Conclusion

This chapter summarized the thesis and discussed potential future research directions in database security. Row permissions, column masks, trusted contexts and holistic database encryption are the key contributions to the database security field made in this thesis. These enhancements equip database systems with the controls necessary to help enterprises effectively implement zero-trust database security and meet security and privacy compliance mandates. Data classification, machine learning and homomorphic encryption are three potential research directions for database security. Data classification would help facilitate the adoption of concepts such as row permissions and column masks by automatically identifying where sensitive data resides so those

constructs can be applied to the appropriate tables and columns. Machine learning can be used to detect unknown threats such as SQL injections and would equip database systems with an additional layer of defense. Homomorphic encryption would remove any privacy concerns when adopting cloud database services and permit organizations to fully benefit from cloud computing.

References

Elmasri, R., Navathe, S. (2010). *Fundamentals of Database Systems*. 6th edition, Addison-Wesley.

Rjaibi, W., Bird, P. (2004). 'A Multi-Purpose Implementation of Mandatory Access Control in Relational Database Management Systems'. *In Proceedings of the International Conference on Very Large Data Bases (VLDB).*

Rjaibi, W. (2004). 'An introduction to multilevel secure relational database management systems'. *In Proceedings of the conference of the Centre for Advanced Studies on Collaborative research (CASCON).*

Gilman, E., Barth, D. (2017). *Zero Trust Networks: Building Secure Systems in Untrusted Networks*. O'Reilly Media.

Voigt, P., von dem Bussche, A. (2017). *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Springer.

Chuvakin, A., Williams, B. (2009). *PCI Compliance: Understand and Implement Effective PCI Data Security Standard Compliance.* Elsevier.

Massachusetts Institute of Technology (MIT) (2019). *Kerberos: The Network Authentication Protocol*. https://web.mit.edu/kerberos/. [Online; accessed 04-January-2020].

Agrawal, R., Bird, P., Grandison, T., Kiernan, J., Logan S., Rjaibi, W. (2005). 'Extending relational database systems to automatically enforce privacy policies'. *In Proceedings of the International Conference on Data Engineering (ICDE).*

Zaytsev, A., Malyuk, A., Miloslavskaya, N. (2017). 'Critical Analysis in the Research Area of Insider Threats'. *In Proceedings of the IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud).*

Ghafir, I., Saleem, J., Hammoudeh, M., Faour, H., Prenosil, V., Jaf, S., Jabbar, S., Baker, T. (2018). 'Security threats to critical infrastructure: the human factor'. *The Journal of Supercomputing*, Volume 74, Issue 10, Springer.

Gaetjen, S., Knox, D., Maroulis, W. (2015). *Oracle Database 12c Security.* McGraw-Hill Education.

Carter, P. (2018). *Securing SQL Server: DBAs defending the database.* Apress.

Vertica (2019). https://www.vertica.com/documentation/vertica/. [Online; accessed 04-January-2020].

Garbus, J. (2015). *SAP ASE 16 / Sybase ASE Administration.* SAP Press.

Chaudhuri, S., Dutta, T., Sudarshan, S. (2007). 'Fine Grained Authorization Through Predicated Grants'. *In Proceedings of the International Conference on Data Engineering (ICDE)*.

Chen, W., Barkai, B., DiPietro, J., Langman, V., Perlov, D., Riah, R., Rozenblit, Y., Santos, A. (2014). *Deployment Guide for Infosphere Guardium.* IBM Redbooks.

Imperva (2019). https://www.imperva.com/. [Online; accessed 04-January-2020].

Walker-Roberts, S., Hammoudeh, M., Dehghantanha, A. (2018). 'A Systematic Review of the Availability and Efficacy of Countermeasures to Internal Threats in Healthcare Critical Infrastructure'. *IEEE Access*, 6, pp.25167-25177.

Walker-Roberts, S., Hammoudeh, M. (2018). 'Artificial Intelligent Agents as Mediators of Trustless Security Systems and Distributed Computing Application'. *In: Parkinson S., Crampton A., Hill R. (eds) Guide to Vulnerability Analysis for Computer Networks and Systems. Computer Communications and Networks*. Springer, Cham.

Goldsteen, A., Kveler, K., Domany, T., Gokhman, I., Rozenberg, B., Farkash, A. (2015). 'Application-Screen Masking: A Hybrid Approach'. *IEEE Software*, Volume 32, Issue 4.

Thanopoulou, A., Carreira, P., Galhardas, H. (2012). 'Benchmarking with TPC-H on Off-the-Shelf Hardware: An Experiments Report'. *In Proceedings of the International Conference on Enterprise Information Systems.*

Alloghani, M., Al-Jumeily, D., Hussain, A., Mustafina, J., Baker, T., Aljaaf, A. (2020). 'Implementation of Machine Learning and Data Mining to Improve Cybersecurity and Limit Vulnerability to Cyber Attacks'. *In: Nature-Inspired Computation in Data Mining and Machine Learning*. Springer, Cham.

Aljawarneh, S., Aldwairi, M., Bani Yassein, M. (2018). 'Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model'. *Journal of Computational Science*, Volume 25, Elsevier.

Aldwairi, M., Alsalman, R. (2012). 'Malurls: a lightweight malicious website classification based on url features'. *Journal of Emerging Technologies in Web Intelligence*, Volume, Issue 2, Academy Publisher.

Benfield, B., Swagerman, R. (2001). 'Encrypting Data Values in DB2 Universal Database'. *IBM DeveloperWorks*.

Boobal G. (2018). 'TDE Tablespace Encryption'. http://www.dba-oracle.com/t_adv_plsql_tde_tablespace.htm. [Online; accessed 04-January-2020].

Zilio, D., Rao, J., Lightstone, S., Lohman G. (2004). 'DB2 Design Advisor: Integrated Automated Physical Database Design'. *In Proceedings of the International Conference on Very Large Data Bases (VLDB).*

IBM (2018). *AIX Encrypted File System (EFS)*. https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/devicemanagement/encrypted_file_system.html. [Online; accessed 04-January-2020].

Microsoft (2018). *Encrypted File System (EFS).* https://docs.microsoft.com/en-us/windows/win32/fileio/file-encryption. [Online; accessed 04-January-2020].

Gemalto (A Thales Company) (2018). Online Product Documentation. https://safenet.gemalto.com/data-encryption/data-center-security/protect-file-encryption-software/. [Online; accessed 04-January-2020].

Dufrasne, B., Brunson, S., Reinhart, A., Tondini, R., Wolf, R. (2016). *IBM DS8880 Data-at-rest Encryption*. IBM Redbooks.

Grover, L. (1996). 'A fast quantum mechanical algorithm for database search'. *In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing,*

Oracle (1992). *Trusted Oracle Administrator's Guide.* Oracle Product Documentation.

Oracle (2019). Online Product Documentation, https://docs.oracle.com/cd/B19306_01/network.102/b14267/intro.htm. [Online; accessed 04-January-2020].

Informix (1993). *Informix OnLine/Secure Administrator's Guide*. Informix Product Documentation.

Rayns, C., Behrends, D., Butler, R., Larsen, K., Lin, M., Yuki, G. (2007). *Securing DB2 and Implementing MLS on z/OS*. IBM Redbooks.

Winnard, K., Biondo, J., Figueiredo, W., Hering, P. (2015). *IBM z/OS V2R2 Security*. IBM Redbooks.

Bertino, E., Castano, S., Ferrari, E. (2001). 'On specifying security policies for web documents with an xml-based language'. *In SACMAT*, pages 57–65.

Bertino, E., Ferrari, E. (2002). 'Secure and selective dissemination of xml documents'. *ACM Trans. Inf. Syst. Secure*., 5(3):290–331.

Halcrow, M. (2007). 'eCryptfs: A Stacked Cryptographic File System'. https://www.linuxjournal.com/article/9400. [Online; accessed 04-January-2020].

Vormetric (A Thales Company) (2018). Online Product Documentation. https://www.thalesesecurity.com/products/data-encryption/vormetric-transparent-encryption. [Online; accessed 04-January-2020].

Bhatti, R., Bertino, E., Ghafoor, A., Joshi, J. (2004). 'Xml-based specification for web services document security'. *In IEEE Computer*, volume 4 of 37, pages 41–49.

Zhang, Z., Rjaibi, W. (2006). 'Inter-node Relationship Labelling: A Fine-Grained XML Access Control Implementation Using Generic Security Labels'. *In Proceedings of the International Conference on Security and Cryptography (SECRYPT).*

Clark, J., DeRose, S. (2006). *Language (XPath) version 1.0*. http://www.w3.org/TR/xpath. [Online; accessed 04-January-2020].

Gartner (2019). https://www.gartner.com/en/newsroom/press-releases/2018-08-15-gartner-forecasts-worldwide-information-security-spending-to-exceed-124-billion-in-2019. [Online; accessed 04-January-2020].

Centrify (2019). Online Product Documentation. https://www.centrify.com/education/what-is-zero-trust-privilege. [Online; accessed 04-January-2020].

Rjaibi, W., Hammoudeh, M. (2019). 'Towards Zero-Trust Database Security Part 1'. *IEEE Future Directions Newsletter: Technology Policy & Ethics*, Issue (September 2019).

Rjaibi, W., Hammoudeh, M. (2019). ‘Towards Zero-Trust Database Security Part 2’. *IEEE Future Directions Newsletter: Technology Policy & Ethics*, Issue (December 2019).

Verizon (2017). *Data Breach Investigations Report*. https://www.knowbe4.com/hubfs/rp_DBIR_2017_Report_execsummary_en_xg.pdf. [Online; accessed 04-January-2020].

Chen, W., Rytir, I., Read, P., Odeh, R. (2008). *DB2 Security and Compliance Solutions for Linux, UNIX, and Windows.* IBM Redbooks.

Rjaibi, W. (2018). ‘Holistic Database Encryption’. *In Proceedings of the International Conference on Security and Cryptography (SECRYPT).*

The Ponemon Institute (2019). https://www.ibm.com/security/data-breach. [Online; accessed 04-January-2020].

Rjaibi, W., Hammoudeh, M. (2020). ‘Enhancing and Simplifying Data Security and Privacy for Multitiered Applications’. *Journal of Parallel and Distributed Computing, Special Issue on Enabling Technologies for Energy Cloud.*

IBM (2019). IBM Security Guardium Analyzer Online Product Documentation. https://www.ibm.com/ca-en/marketplace/guardium-analyzer. [Online; accessed 04-January-2020].

Adir, A., Aharoni, E., Greenberg, L., Miroshnikov, R., Rozenberg, B. Sofer, O. (2017). ‘Cyber Security Event Detection’. *US Patent 10397259*.

Popa, R., Redfield, C., Zeldovich, N., Balakrishnan, H. (2011). ‘CryptDB: Protecting Confidentiality with Encrypted Query Processing’. *In Proceedings of the ACM Symposium on Operating Systems Principles (SOSP).*

Chandra, S., Paira, S., Alam, S., Sanyal, G. (2014). ‘A comparative survey of Symmetric and Asymmetric Key Cryptography’. *In Proceedings of the International Conference on Electronics, Communication and Computational Engineering.*

Shor, P. (1997). ‘Polynomial time algorithms for prime factorization and discrete logarithms on a quantum’. *In SIAM J. Sci. Statist. 26 (1997)*.

Bruno N., Kwon, Y., Wu, M. (2014). ‘Advanced Join Strategies for Large-Scale Distributed Computation’. *In Proceedings of the Very Large Data Bases (VLDB)*

Endowment, Vol. 7, No. 13.

Balkesen, C., Alonso, G., Teubner, J., Ozsu, M. (2013). 'Multi-Core, Main-Memory Joins: Sort vs. Hash Revisited'. *In Proceedings of the Very Large Data Bases (VLDB) Endowment*, Vol. 7, No. 1.

Bruni, P., Harrison, K., Oldham, G., Pedersen, L., Tino, G. (2007). *DB2 9 for z/OS Performance Topics.* IBM Redbooks.

Appendix A: Fine-Grained Authorization Portfolio

Table A.1 – Research Papers

ID	Publication	Key Contributions
1	**Enhancing and Simplifying Data Security and Privacy for Multitiered Applications** *Journal of Parallel and Distributed Systems, Special issue on Enabling Technologies for Energy Cloud* *(Also, Chapter 3 of this thesis)*	- Design of a holistic fine-grained database authorization solution which allows organizations to reduce the complexity of their applications and improve overall database security. - Enable organizations to adhere to zero-trust security. - Implementation of the solution in IBM DB2 for Linux, Unix and Windows, IBM DB2 for z/OS and IBM for DB2 for iSeries.
2	**Extending Relational Database Systems to Automatically Enforce Privacy Policies** *International conference on Data Engineering (ICDE)*	- Design of a solution which extends database systems to be able to automatically enforce privacy policies. - Enable organizations to meet privacy requirements for data stored in database systems.

Extending Relational Database Systems to Automatically Enforce Privacy Policies

Rakesh Agrawal† Paul Bird‡ Tyrone Grandison† Jerry Kiernan† Scott Logan‡ Walid Rjaibi‡

† IBM Almaden Research Center
650 Harry Road, San Jose, CA, USA
{ragrawal, tyroneg, jkiernan}@us.ibm.com

‡ IBM Toronto Lab
8200 Warden Ave.,Markham, ON, Canada
{pbird, silogan, wrjaibi}@ca.ibm.com

Abstract

Databases are at the core of successful businesses. Due to the voluminous stores of personal data being held by companies today, preserving privacy has become a crucial requirement for operating a business. This paper proposes how current relational database management systems can be transformed into their privacy-preserving equivalents. Specifically, we present language constructs and implementation design for fine-grained access control to realize this goal.

1. Introduction

The pervasive use of computing technology and the increased reliance on information systems have created a heightened awareness and concern about the storage and use of private information. This worldwide phenomenon has ushered in a plethora of privacy-related guidelines and legislations, e.g. the OECD Privacy Guidelines in Europe, the Canadian Privacy Act, the Australian Privacy Amendment Act, the Japanese Privacy Code, the Health Insurance Portability and Accountability Act (HIPAA), and Gramm-Leach-Bliley Consumer Privacy Rule. Compliance with these legislation has become an important corporate concern. The current methods employed to address the disclosure compliance problem involve training individuals to be cognizant of the various regulations and changing organizational processes and procedures. However, these approaches are only a partial solution and need to be augmented with technology support.

We present constructs for imbuing relational database systems with fine grained access control (FGAC) and show how they can be used to enforce disclosure control enunciated in the vision for Hippocratic databases [1]. These constructs have been designed to be integrated with the rest of the infrastructure of a relational database system. We also discuss the implementation of the proposed FGAC constructs, building upon the ideas from [6]. Finally, we show how privacy policies written in a higher-level specification language such as P3P [3] can be algorithmically translated into the proposed constructs.

The users of relational databases are requiring that an FGAC implementation meets the following desiderata:

- The implementation must solve the problem within the database itself without application changes or application awareness of the implementation.
- The implementation must ensure that all users of the data are covered, regardless of how the data is accessed.
- The implementation must minimize the complexity and maintenance of the FGAC policies.
- The implementation must provide the ability to control access to rows, columns, or cells as desired.

Traditional methods of database access control have relied upon the use of statically defined views, which are logical constructs imposed over database tables that can alter or restrict the data seen by a user. Using predefined views as the method for FGAC works well only when the number of different restrictions is few or the granularity of the restrictions is such that it affects large, easily identified groups of users. When these conditions are not true, view definitions may become complex in an effort to accommodate all the restrictions in one view. This complexity can strain system limits and can make maintenance of the views difficult.

Consider the use of a large number of views, each one implementing restrictions for a specific set of users. One issue that arises immediately is how to correctly route user requests to the view that is appropriate to them. Often, the solution chosen is to resolve the request in the application,

not in the database. Moreover, if a user can bypass the view when accessing data, for example by having direct access to the underlying tables, then the restrictions are not enforced.

Given the shortcomings of the traditional methods of implementing FGAC, some database vendors have proposed solutions that do not rely on the use of views to control access to tabular data. For instance, the Oracle Virtual Private Database [5, 7] solution allows users to define a security policy, which is a function written in PL/SQL that returns a string representing a predicate, and to attach the security policy to a table. When that table is accessed, the security policy is automatically enforced. Sybase Row Level Access Control [9] allows users to define access rules that apply restrictions to retrieved data. The related work section further discusses the Oracle and Sybase approaches. Microsoft SQL Server primarily supports traditional view based access control, though they have a feature called row level permissions. However, row level permissions seem to be usable only with table hierarchies. In DB2, support for FGAC is currently provided through traditional mechanisms based on views, triggers and special registers.

The remainder of this paper is organized as follows. Section 2 proposes FGAC constructs that allow restrictions to be expressed on database accesses. Aside from row and column level restrictions that respectively restrict the set of rows and columns of a table, cell level restrictions can be specified to limit access to specific fields of a row. Section 3 describes how restrictions expressed in terms of the proposed constructs can be enforced using dynamic views. Section 4 presents an algorithm for translating a P3P privacy policy into the proposed FGAC constructs. Section 5 discusses related work, and Section 6 presents concluding remarks. Appendix A argues for extending the functionality of current relational database systems with cell level access control.

2. Language Constructs

We provide constructs that allow restrictions to be specified on access to data in a table at the level of a row, a column, or a cell (i.e., individual column-row intersections). Privacy policies specified in high-level languages such as P3P can be translated into these constructs, or one could specify the policy directly using these constructs.

The proposed facility is complimentary to the current table level authorization mechanisms provided by commercial database systems using the **grant** command [2]. While grant controls whether a user can access a table at all, the proposed constructs define the subset of the data within a table that the user is allowed to access. Conceptually, a restriction defines a view of the table in which inaccessible data has been replaced by null values. As discussed in [6], it is possible to use either "table semantics" or "query semantics". With query semantics, if all the values in a row are hidden by a restriction, then the row is omitted altogether from the view. With table semantics, the row would instead be retained.

```
create restriction restriction-name
on table-x
for auth-name-1 [ except auth-name-2]
( (    (to columns column-name-list)
     | (to rows [ where search-condition ] )
     | (to cells (column-name-list [ where search-condition ] )+ )
  )
  [ for purpose purpose-list ]
  [ for recipient recipient-list ]
)+
command-restriction
```

Figure 1. Fine grained restriction syntax

Figure 1 gives the syntax of a fine grained restriction command. It states that those in auth-name-1 except those in auth-name-2 are allowed only restricted access to table-x. The keywords **public** (i.e., all users), **group**, **role**, and **user** can be used to qualify the authorized names. Table-x can be any table expression.

A restriction can be specified at the level of a column (Section 2.1), a row (Section 2.2), or a cell (Section 2.3). More than one restriction can be specified on a table for the same user (Section 2.4).

A restriction may additionally specify purposes and/or recipients [1, 3, 6] for which the access is allowed. If no purpose or recipient is specified, then the restriction applies to all purposes and recipients respectively. If a purpose or recipient is specified, the user's access is limited to only the specified purpose-recipient combinations.

Akin to the database system variable **user** that can be referenced in queries and returns the id of the user issuing the query, the new system variables **purpose** and **recipient** return the list of purposes and recipients from the current query context [6]. These values in turn determine the restrictions for the current query.

The command-restriction that appears as the last element of the syntax has the following form and states that access can be restricted to any combination of select, delete, insert, or update commands:

restricting access to (**all** | (**select** | **delete** | **insert** | **update**)+)

The discussion below will use, for illustration, the Customer table with the following schema: Customer (id **integer**, name **char**(32), phone **char**(32)).

2.1 Column Restriction

A column restriction specifies a subset of the columns in table-x that auth-name-1 is allowed to access. The following restriction, named r1, ensures that only the id column of Customer is accessed by any database user:

create restriction r1
on Customer
for public
to columns id
restricting access to all

The restriction r2 below ensures that members of the account group and user Bob have only select access to columns name and phone.

create restriction r2
on Customer
for group acct, **user** Bob
to columns name, phone
restricting access to select

2.2 Row Restriction

A row restriction gives the subset of rows in table-x that auth-name-1 is allowed to access. This subset is specified using a search-condition over table-x. The restriction r3 below ensures that every access to Customer is qualified by the predicate, name = **user**.

create restriction r3
on Customer
for public
to rows where name = **user**
restricting access to all

If user Bob issues **select** * **from** Customer, he would see id, name and phone for those rows where name equaled Bob.

2.3 Cell Restriction

A cell restriction defines the row-column intersections that auth-name-1 is allowed to access. It is possible to specify multiple column-name lists, each possibly annotated with a search-condition. A search-condition is a correlated subquery with an implicit correlation variable t defined over the tuples of table-x. Access to the columns in column-name-list for each individual row identified by t is conditionally granted depending upon the result of the search condition. If no search-condition is given, then access is granted to all column values in column-name-list in table-x. If the search condition ignores correlation variable, then access is granted or denied to all columns values in column-name-list in table-x, depending upon the result of the search-condition.

The following is an example of a cell restriction used to enforce individual user's privacy preferences expressed as opt-in/out choices. Assume that for the purpose of marketing, Bob is allowed to see name, but his access to phone is allowed only if the user has opted-in to revealing her phone number.

create restriction r4
on Customer **for user** Bob,
to cells name,
(phone **where exists** (
select 1
from SysCat.Choices_Customer c
where c.ID = Customer.ID **and** c.C1 = 1))
for purpose marketing
for recipient others
restricting access to select

The above restriction specifies cell restrictions for two column-name-lists: The first list contains the name column, and the second contains the phone column. The restriction allows Bob access to name, only if the variable **purpose** includes marketing, and **recipient** includes others. Otherwise, all values of the name column will be null for Bob.

The second list of columns has a search-condition associated with it since access to phone is dependent upon individual user choices. The search-condition comprises an existential subquery that uses the implicit correlation variable Customer. For each row in Customer, the subquery verifies, using the SysCat.Choices_Customer table that stores individual opt-in/out choices, whether the user has opted-in for the disclosure of her phone number (represented by a column value of 1).

2.4 Combining Multiple Restrictions

If multiple restrictions have been defined for a user u and a table T, then u's access to T is governed by the combination of these restrictions.

Assume initially that a user associates with a query a single purpose and a single recipient. We consider two design choices for combining multiple restrictions:

- *Intersection* — User u is allowed access to data defined by the intersection of all applicable restrictions. The details are shown in Table 1.
- *Union* — User u is allowed access to data defined by the union of all applicable restrictions. The details are shown in Table 2.

If the commands specified in the command-restriction clauses of the restrictions being combined are different, they

	row	column	cell
row	The search-conditions of individual row restrictions are and'ed together to define the intersection of rows accessible to a user.	The row restriction limits the rows accessible to the user. The column restriction further limits the columns within the rows accessible to the user.	The row restriction limits the rows accessible to the user. Within each row, the cell restriction further limits the access to the cells that qualify the cells' search-condition.
column		The user's access is limited to those columns that appear in both of the column restrictions.	Column and cell restrictions intersect to limit access to only those columns that appear in both the restrictions. In addition, the cell restriction's search-condition further limits accessible cells within a column.
cell			The search-conditions are and'ed together and the user is allowed access to a cell if the composite condition is satisfied for the cell. The value of the composite condition for a cell that does not appear in both the restrictions is false.

Table 1. Combining restrictions with intersection

	row	column	cell
row	The search-conditions of individual row restrictions are or'ed together to define the union of rows accessible to a user.	The user is given access to all the cells for any row that satisfies the row restriction. Additionally, the user is allowed access to all the cells in any of the columns that satisfies the column restriction, irrespective of whether the corresponding rows satisfy the row restriction.	The user is given access to all the cells in any of the rows that satisfy the row restriction. Additionally, the user is allowed access to all other cells that satisfy the cell restriction's search-condition, irrespective of whether the corresponding rows satisfy the row restriction.
column		The user is allowed access to a column if it appears in either of the two column restrictions.	The user is given access to all the cells in any column appearing in the column restriction, regardless of whether the cell restriction is satisfied for these cells. For cells in a column for which the column restriction does not apply, access is given if the cell restriction is satisfied.
cell			The search-conditions are or'ed together and the user is allowed access to a cell if the composite condition is satisfied for the cell.

Table 2. Combining restrictions with union

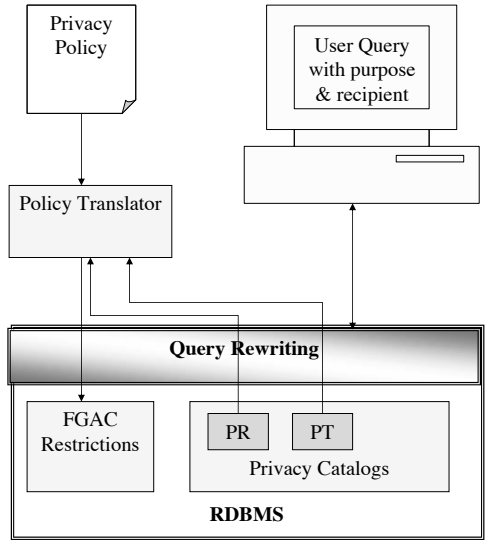


Figure 2. Implementation architecture

are respectively and'ed or or'ed depending upon the choice of intersection or union semantics.

Multiple restrictions can be combined in any order, both with intersection and union semantics. With the intersection semantics, the user's access to data decreases as additional restrictions are applied. Conversely, with union semantics, access to data increases as additional restrictions are applied.

We prefer intersection semantics over union since additional restrictions should intuitively decrease a user's access to information.[1]

Finally, if a query is annotated with multiple purpose-recipient pairs, instead of a single pair, then restrictions governing access to any of the pairs become relevant for the query. These restrictions are then combined as above. Note that once a user's access to a table has been restricted, the user can only access the data allowed for the purposes and recipients specified in the restrictions.

3. Implementation

We next present a design for implementing the proposed constructs, building upon the ideas presented in [1, 6]. In this and the remainder of the paper, we focus on cell restrictions limited to select statement access. Figure 2 shows the overview of the design. The policy translator accepts a privacy policy (written in, for example, P3P) and metadata stored in privacy catalogs and generates cell restrictions that implement the policy. The schema of the privacy metadata catalogs shown in Figure 2 used to drive the translation of P3P privacy policies into cell level restrictions are given below.

PR (purp-recip **char**(18),
p3ptype **char**(32),
choice_tabname **char**(32),
choice_colname **char**(32))

PT (p3ptype **char** (32), tabname **char**(32), colname **char**(32))

Table PR stores, for each purpose, recipient and p3p data type pair, the (table name-column name) pair that records individual user opt-in/out choice, should any choice be available for that combination. Table PT stores, for each P3P data type, the table names and column names which store values of these P3P types.

Figure 3 gives the algorithm used for enforcing the fine grain restrictions. For ease of exposition, we assume there is a single purpose-recipient pair associated with a query and there is at most a single restriction which is relevant for the query. The enforcement algorithm combines the restrictions relevant to individual queries annotated with purpose and recipient information and transforms the user's query into an equivalent query over a dynamic view that implements the restriction.

In detail, Line 1 iterates over each table reference t in a query Q. Line 2 accesses metadata to determine if there is a restriction r governing the usage of t by user u who is submitting the query Q. If no such restriction exists, then t remains unmodified in Q. Otherwise, Lines 3 and 4 replace each reference to table t in query Q with a reference to a dynamic view v.

The generation of the dynamic view v is implemented in Lines 5 through 25. The view v is a select statement which conditionally projects each column $c \in t$. Line 7 searches for a column reference to $c \in r$. If no such reference exists with the purpose/recipient of query Q, then the user u is not allowed access to c and Line 8 thus projects a null value for all values of c. Otherwise, Line 10 searches for a where clause associated with $c \in r$. If no such clause exists, then u is granted unconditional access to c. Otherwise, Line 15 outputs the condition of the where clause into a SQL case statement which verifies the condition before outputting the value of c (on Line 18). If the condition is false, access to the column value is denied and Line 19 outputs a null value for c.

4. Translating Privacy Policies

It is expected that the privacy policies will likely be written in some high-level policy language. The following illus-

[1] It is conceivable to use mixed modes for combining restrictions. For example, intersection could be used to combine multiple row restrictions while union could be used to combine multiple column or cell restrictions. However, the semantics of such combinations can become quite complex as the restriction imposed by a combination may no longer be order-independent.

```
1  for each table reference t in query Q do begin
2    if (exists a restriction r pertaining to t for Q) then begin
3      create a dynamic view v ∈ Q over t
4      replace each reference to t ∈ Q with a reference to v ∈ Q

       // create the dynamic view v using
       // the following print statements
       //
5      print "select"
6      for each column c ∈ t do begin
         // c_p, c_r are the purposes, recipients
         // of column c in restriction r
         // Q_p, Q_r are the purpose, recipient of query Q
         //
7        if (c ∉ r|Q_p ∈ c_p ∧ Q_r ∈ c_r
           // c isn't included in the restriction r
           // access to c is thus prohibited
           //
8          print "null"
9        else begin
           // The whereClause function returns
           // the predicate associated with c
           // that is specified in the restriction
           //
10         let w = whereClause(c)
11         if w = null then
             // There is no "where" condition
             // governing the use of c ∈ r, thus access
             // to all column values is granted unconditionally
             //
12           print c.colname
13         else begin
             // Implement the "where" condition
             // using a SQL case statement to grant
             // only conditional access to the column c
             //
14           print "case when exists ("
15           print w.condition
16           print ")"
17           print "then"
18           print c.colname
19           print "else null end as"
20           print c.colname
21         end
22       end
23     end
24     print "from"
25     print t.tablename
26   end
```

Figure 3. Algorithm for enforcing fine grained cell level restrictions using a Hippocratic database system

trates the basic syntax of the P3P policy specification language [3].

```
<POLICIES> ...
 <POLICY name = "Policy_Name1"  > ...
  <STATEMENT>
   ...
   <PURPOSE>
     stated-purpose
      [ required = ("always"|"opt-in"|"opt-out") ]
   </PURPOSE>
   <RECIPIENT>
     stated-recip
      [ required = ("always"|"opt-in"|"opt-out") ]
   </RECIPIENT>
   <RETENTION>  retention_val  </RETENTION>
   <DATA GROUP>
    <DATA ref = data-ref-val>
     ...
    </DATA GROUP>
  </STATEMENT>
 </POLICY>
 <POLICY>
  ...
 </POLICY>
 ...
</POLICIES>
```

The process of transforming a policy like the one above into fine grained restrictions involves: (1) parsing the policy to extract the list of statements, (2) mapping data abstractions into their implementation specific equivalents, e.g. in the above specification this would mean mapping data-ref-val to its corresponding table name(s) and column name(s), (3) structuring the choice tables which record individual user opt-in/out choices (in some cases, this may not be necessary since there may be no such choices), and (4) generating the restriction statements. Assuming that data-ref-val maps to columns A and B of table T, the above abstract specification would lead to the following restriction being constructed:

create restriction Policy_Name1
on T
for public
to cells A,B
  [**where** opt-in-out-conditions]
  for purpose stated-purpose
  for recipient stated-recip
restricting access to select

Figure 4 is a detailed example of a privacy policy, for a fictional Healthcare provider.

The metadata contains the information needed to associate "#personal" (personal information) and "#medical" (medical information) with database tables which store this information. Personal information maps to the name, SSN, address, email and DOB fields of the Patients table, while medical information maps to the xray, pharmacy, family, appointment and lifestyle fields of the Patients table. Thus,

```
...
<!-- Statement1 -->
<STATEMENT>
 <CONSEQUENCE>
  Encodes that personal and medical information
  can be accessed for emergency purposes
  by ourselves
 </CONSEQUENCE>
 <PURPOSE>
  <other-purpose>
    Emergency
  </other-purpose>
 </PURPOSE>
 <RECIPIENT><ours/></RECIPIENT>
 <RETENTION><stated-purpose/></RETENTION>
 <DATA-GROUP>
  <DATA ref = "#personal"/>
  <DATA ref = "#medical">
   <CATEGORIES>
    <health/>
   </CATEGORIES>
  </DATA>
 </DATA-GROUP>
</STATEMENT>

<!-- Statement2 -->
 <STATEMENT>
  <CONSEQUENCE>
   Encodes that we and drug companies
   with the same data usage policies
   can access personal and medical information
   for new_drug_research on an opt-out basis
  </CONSEQUENCE>
  <PURPOSE><develop/></PURPOSE>
  <RECIPIENT>
   <ours required="opt-out"/>
   <same required="opt-out"/>
  </RECIPIENT>
  <RETENTION><stated-purpose/></RETENTION>
  <DATA-GROUP>
   <DATA ref = "#personal"/>
   <DATA ref = "#medical">
    <CATEGORIES>
     <health/>
    </CATEGORIES>
   </DATA>
  </DATA-GROUP>
 </STATEMENT>
 ...
```

Figure 4. A sample privacy policy written for a health care provider

```
create restriction Statement1
on Patients
for public
to cells          Name, SSN, Address, Email, DOB,
                  XRay, Pharmacy, Family,
                  Appointment, Lifestyle
    purpose Emergency
    recipient ours
restricting access to select

create restriction Statement2.1
on Patients
for public
to cells          Name, SSN, Address, Email, DOB,
                  XRay, Pharmacy, Family,
                  Appointment, Lifestyle
      where
        exists (
          select 1
          from SysCat.Choices_Patients cp
          where cp.ID = Patients.ID
            and cp.C1 = 1 )
    for purpose develop
    for recipient ours
restricting access to select

create restriction Statement2.2
on Patients
for public
to cells          Name, SSN, Address, Email, DOB,
                  XRay, Pharmacy, Family,
                  Appointment, Lifestyle
      where
        exists (
          select 1
          from SysCat.Choices_Patients cp
          where cp.ID = Patients.ID
            and cp.C2 = 1 )
    for purpose develop
    for recipient same
restricting access to select
```

Figure 5. Translation of the privacy policy in Figure 4 into fine grained cell level restrictions

the P3P healthcare policy given in Figure 4 is translated into the restrictions given in Figure 5.

For simplicity, the restrictions in Figure 5 assume that all data types in a P3P statement are contained in a single table.

The Choices_Patients table is created by the database administrator to record individual opt-in/out decisions described in the privacy policy. In Figure 5, C1 represents the choice to allow Drug_Research to see personal and medical data if the drug research is being conducted by the healthcare company itself. Choice C2 is the option to allow usage of the personal and medical data for drug research by other healthcare companies having the same privacy policy as this company. The example illustrates the basic steps involved in the translation process.

Figure 6 gives the pseudo-code showing the steps involved in transforming P3P policy into our proposed constructs. A unique restriction name, needed for the command, is generated on Line 5. Line 7 uses the **mapP3PStatementToTable** function to recover the table name which stores the information described by the data types in the P3P statement. This metadata has been populated by the database administrator. On Line 8, the the restriction is set to **public** to apply to all users. Line 10 uses the **mapP3PDataTypeToColumns** function to retrieve the column names that store information described by the P3P data types in the statement. Again, this information has been prepared and supplied by the database administrator and stored in metadata tables.

The function **mapP3PPurposeToChoiceTable** accepts a statement id and returns the table storing individual user choices for this statement. The function **mapP3PPurposeToChoiceColumn** accepts a statement-purpose pair and returns the column in the choice table which records the corresponding users' choices. Both these functions are driven from metadata.

```
1  for each statement s in policy do begin
2    for each purpose p in s do begin
3      for each recipient r in s do begin

4        print "create restriction "
5        print generate-unique-restriction-name()
6        print " on table "
7        print mapP3PStatementToTable(s)
8        print " for public "
9        print " to cells "
10       print mapP3PDataTypeToColumns(s)

11       if (p.required != always) then
12         print "where exists (select 1 from "
13               + mapP3PPurposeToChoiceTable(s)
14               + " p where p.ID = "+ mapP3PStatementToTable(s) +".ID
15               and "+ mapP3PPurposeToChoiceColumn(s,p) + "= 1))"

16       if (r.required != always) then
17         print "and exists (select 1 from "
18               + mapP3PRecipientToChoiceTable(s)
19               + " r where r.ID = "+ mapP3PStatementToTable(s) +".ID"
20               + "and "+ mapP3PRecipientToChoiceColumn(s, r) + "= 1))"
21       print "for purpose" + p.name
22       print "for recipient" + r.name
23     end
24   end
25 end
26 print "restricting access to select"
```

Figure 6. Algorithm for translating a P3P privacy policy into fine grained cell level restrictions

5. Related Work

5.1 Oracle

Oracle has introduced a fine-grained access control capability via their security policy concept [5, 7] which, once defined on a table or view, modifies any future query against that table by adding a predicate into the query. In essence, they have allowed row restrictions traditionally handled by views to be dynamically added to queries [8].

The fundamental difference between the Oracle approach and the one in this paper is that Oracle modifies the query by adding predicates while the approach in this paper leaves the query alone and effectively modifies the table being accessed by injecting a dynamically created view of the table between the query and the target table.

The Oracle approach shares the following advantages with our design:

- It is pervasive to all users of the table.
- It does not require application modification.
- It does not require a large number of statically defined views.

Its primary disadvantages are:

- It requires user programming of a strictly defined "predicate producing" procedure in order to implement a security policy.
- It does not address column or cell restrictions.

5.2 Sybase

Sybase Adaptive Server version 12.5 has introduced a feature called row level access control [9] that enables the database owner or table owner to restrict access to a table's

rows by defining access rules and binding those rules to the table. Access to data can be further controlled by setting application contexts and creating login triggers.

Access rules apply restrictions to retrieved data, enforced on select, update and delete operations. Adaptive Server enforces the access rules on all columns that are read by a query, even if the columns are not included in the select list. Using access rules is similar to using views, or using an ad hoc query with where clauses. The query is compiled and optimized after the access rules are attached, so it does not cause performance degradation. Access rules provide a virtual view of the table data, the view depending on the specific access rules bound to the columns.

Our proposal differs from the Sybase row level access control solution as follows:

- It allows restrictions to be defined on columns and cells in addition to rows.
- A restriction can contain as many predicates as desired and this is done in a single statement (i.e., create restriction). Sybase would need to create a separate access rule for each predicate, and'ing them, and then binding them to the appropriate columns.

6. Conclusion

Databases of the future must ensure the privacy of the data subjects that they store information on. The security functionality offered by current commercial database products is not adequate to enforce privacy compliance. The main contributions of this paper are:

- Language constructs for specifying restrictions at the level of a row, a column, or a cell that integrate well with the rest of the relational database infrastructure.
- Semantics of combining multiple restrictions.
- Design for implementing the proposed constructs.
- Algorithm for translating a P3P privacy policy into the proposed constructs.

Our fond hope is that this paper will serve to create dialog on the right functionality that the database systems must support and the efficient ways of its implementation.

Acknowledgments We wish to thank Alvin Cheung for useful comments on the paper.

References

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *28th Int'l Conference on Very Large Databases*, Hong Kong, China, August 2002.

[2] D. Chamberlin. *A Complete Guide to DB2 Universal Database*. Morgan Kaufmann, 1998.

[3] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*. W3C Recommendation, April 2002.

[4] US Department of Health and Human Services. http://www.hhs.gov/ocr/hipaa.

[5] T. Kyte. Fine-grained access control. Technical report, Oracle Corporation, 1999.

[6] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt. Limiting disclosure in Hippocratic databases. In *30th Int'l Conf. on Very Large Data Bases*, Toronto, Canada, August 2004.

[7] A. Nanda and D. K. Burleson. *Oracle Privacy Security Auditing*. Rampant, 2003.

[8] M. Stonebraker and E. Wong. Access control in a relational data base management system by query modification. In *ACM/CSC-ER*, 1974.

[9] Sybase. *Sybase Adaptive Server Enterprise 12.5, System Administration Guide, Row Level Access Control*. http://sybooks.sybase.com/onlinebooks.

ID	Name	HomePhone	WorkPhone	Salary
1	Alicia Campbell	408-418-5198	408-419-9111	10,000
2	Bob Bobbett	408-418-5198	408-419-9112	20,000
3	Carl Abrahams	408-333-6633	408-419-9113	30,000
4	Dan Charmer	408-432-8644	408-419-9114	40,000
5	Ellen Generous	408-555-1235	408-419-9115	50,000

Table 3. Table of BlueCo's clients

Name	HomePhone	OfficePhone
Alicia Campbell	-	408-419-9111
Bob Bobbett	408-418-5198	-
Carl Abrahams	408-333-6633	408-419-9113

Table 4. Cell level enforcement

A. The Case for Cell Level Enforcement

Compliance with current privacy legislation mandates that the user's consent be obtained for the use/disclosure of the personal information stored about them. Row or column level restriction are not adequate for modeling scenarios where individuals may make opt-in/out choices with different aspects of their information. To achieve this goal of minimal disclosure while allowing useful tasks to be performed on relevant information, cell level enforcement is key. A similar case for cell level enforcement has been made in [6].

Consider a scenario requiring adherence to the HIPAA regulation [4]. BlueCo is a healthcare provider that stores personal data on individuals who enroll in its plans. BlueCo has affiliations with a number of hospitals, research institutions, and marketing companies. Under HIPAA, any individually identifiable healthcare information held or transmitted by BlueCo is considered protected information. For any use or disclosure of protected health information that is not for treatment, payment, or health care operation and that is not otherwise permitted (e.g. law enforcement), BlueCo must get the data subject's consent.

Assume a simplified version of BlueCo's database given in Table 3. ResearchCo is an epidemiological research institute that periodically harvests BlueCo's data. Under HIPAA, all clients must give their consent for release of their home and office numbers.

Alicia Campbell opts out of having her home phone number, but does not mind if BlueCo discloses her office number. Suppose John Seeker, a researcher at ResearchCo issues the following query:

select name, homephone, officephone
from clients **where** salary $\leq$ 30000

Given the choices that Alicia has made, only her name and office phone number should be displayed as shown in Table 4.

Name	HomePhone	OfficePhone
Carl Abrahams	408-333-6633	408-419-9113

Table 5. Row level enforcement

Database systems employing row level controls restrict disclosure to all information in a particular row, when a restriction is only on particular columns in that row. Thus, using conventional row level controls, the results for the query are those shown in Table 5. Both Alicia and Bob are no longer present in the result, even though they have agreed that one of their two phone numbers can be disclosed.

This simple example illustrates the inadequacy of row level restrictions. Similar arguments can be made for column level restrictions. They are not flexible enough to allow disclosure of non-sensitive data and suppression of sensitive data on a subject by subject basis.

Table A.2 – Granted Patents

ID	Publication	Key Contributions
1	**Method and System for Using Fine-Grained Access Control (FGAC) to Control Access to Data in a Database** *US Patent US8,234.299B2*	This patent is the foundation for the row permission and column mask concepts discussed in the core publication #1 in table A.1 above.
2	**Method for Establishing a Trusted Relationship Between a Data Server and a Middleware Server** *US Patent US 7,647,626B2*	This patent is the foundation for the trusted context concept discussed in the core publication #1 in table A.1 above.
3	**Access Control for Elements in a Database Object** *US Patent US7,865,521B2*	This patent is the foundation for the table restriction concept discussed in publication #2 in table A.1 above.
4	**Extending Relational Database Systems to Automatically Enforce Privacy Policies** *US Patent US 7,243,097 B1*	This patent is the foundation for the method to translate privacy policies into table restrictions discussed in publication #2 in table A.1 above.

US008234299B2

(12) **United States Patent**
Bird et al.

(10) **Patent No.: US 8,234,299 B2**
(45) **Date of Patent: Jul. 31, 2012**

(54) **METHOD AND SYSTEM FOR USING FINE-GRAINED ACCESS CONTROL (FGAC) TO CONTROL ACCESS TO DATA IN A DATABASE**

(75) Inventors: **Paul Miller Bird**, Markham (CA); **Yao-Ching Stephen Chen**, Saratoga (CA); **George Gerald Kiernan**, San Jose, CA (US); **Scott Ian Logan**, Don Mills (CA); **Allen William Luniewski**, Cupertino, CA (US); **Walid Rjaibi**, Markham, CA (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 472 days.

(21) Appl. No.: **12/013,253**

(22) Filed: **Jan. 11, 2008**

(65) **Prior Publication Data**

US 2009/0182747 A1 Jul. 16, 2009

(51) **Int. Cl.**
G06F 7/00 (2006.01)
G06F 17/30 (2006.01)

(52) **U.S. Cl.** **707/781**

(58) **Field of Classification Search** 707/9, 694, 707/713, 781, 999.009
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,987,455	A *	11/1999	Cochrane et al.	1/1
6,085,191	A *	7/2000	Fisher et al.	707/737
6,212,511	B1 *	4/2001	Fisher et al.	1/1
6,236,996	B1	5/2001	Bapat et al.	
6,487,552	B1 *	11/2002	Lei et al.	1/1
6,813,617	B2 *	11/2004	Wong et al.	1/1
7,483,896	B2 *	1/2009	Johnson	1/1
2002/0016924	A1 *	2/2002	Shah et al.	713/200
2003/0014394	A1 *	1/2003	Fujiwara et al.	707/3
2003/0236781	A1 *	12/2003	Lei et al.	707/3
2004/0044655	A1	3/2004	Cotner et al.	
2004/0205355	A1	10/2004	Boozer et al.	
2005/0144176	A1 *	6/2005	Lei et al.	707/100
2005/0177570	A1 *	8/2005	Dutta et al.	707/9
2005/0246338	A1 *	11/2005	Bird	707/9
2005/0289342	A1 *	12/2005	Needham et al.	713/169
2006/0020581	A1 *	1/2006	Dettinger et al.	707/3
2006/0059567	A1 *	3/2006	Bird et al.	726/27
2008/0010233	A1 *	1/2008	Sack et al.	707/1
2008/0071785	A1 *	3/2008	Kabra et al.	707/9

(Continued)

OTHER PUBLICATIONS

Chaudhuri et al., Fine Grained Authorization Through Predicated Grants, Apr. 2007, IEEE Xplore, pp. 1174-1183.*

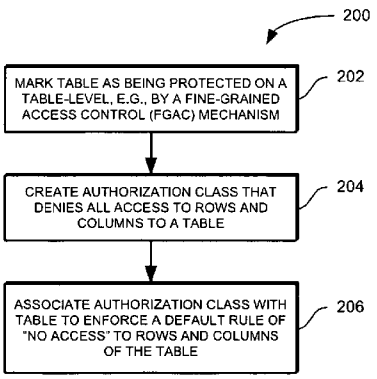
(Continued)

Primary Examiner — Wilson Lee
Assistant Examiner — Jessica N Le
(74) *Attorney, Agent, or Firm* — Sawyer Law Group, P.C.

(57) **ABSTRACT**

A method and system for controlling access to data stored in a table of a database are provided. The method includes marking the table of the database as being protected with fine-grained access control (FGAC), creating a system authorization class for the table of the database, the system authorization class having a default row authorization that prevents access to all rows in the table, the system authorization class being unmodifiable, creating a user authorization class for the table of the database, the user authorization class having a default row authorization that prevents access to all rows in the table, the user authorization class being modifiable, and associating the system authorization class and the user authorization class with the table of the database.

18 Claims, 2 Drawing Sheets



U.S. PATENT DOCUMENTS

2008/0162402	A1*	7/2008	Holmes et al.	707/1
2008/0313134	A1*	12/2008	Lei	707/2
2008/0319999	A1*	12/2008	Simpson et al.	707/9

OTHER PUBLICATIONS

Leicester, J. M., "VPDand Columnar FGAC" Oramoss Oracle, http://72.14.203.104/search?q=cache: yPsSX99vWwAJ:oramossoracle.blogspot.com/+database+FGAC+%22column-level+security22&hl=en&gl=us&ct=clnk&cd=3, Jan. 8, 2006, 4 Pages.

Burleson Consulting, "Oracle Virtual Private Database Policy (VPD) Tips", http://www.dba-oracle.com/art_builder_vpd.htm, Oracle Virtual Private Database VPD with RLS and FGAC, Aug. 25, 2003, 5 Pages.

* cited by examiner

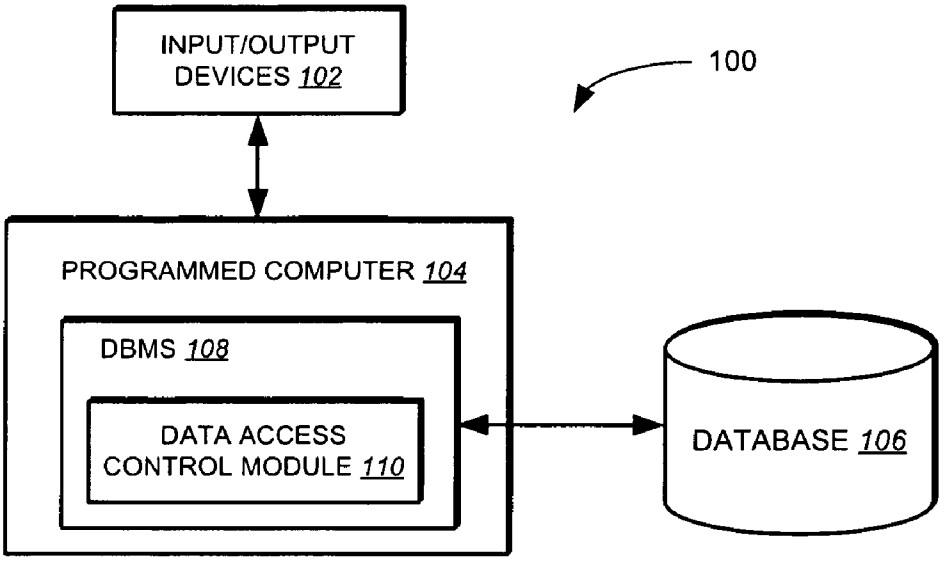

INPUT/OUTPUT DEVICES 102
100
PROGRAMMED COMPUTER 104
DBMS 108
DATA ACCESS CONTROL MODULE 110
DATABASE 106

FIG. 1

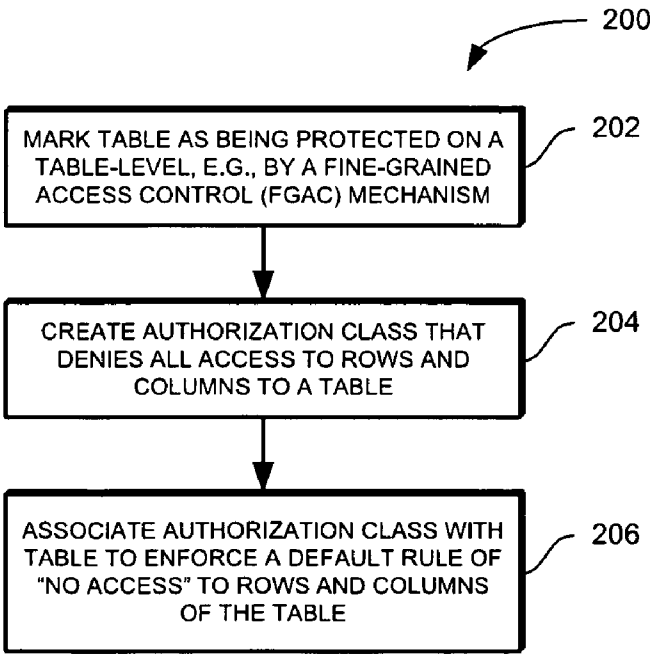

200
MARK TABLE AS BEING PROTECTED ON A TABLE-LEVEL, E.G., BY A FINE-GRAINED ACCESS CONTROL (FGAC) MECHANISM
202
CREATE AUTHORIZATION CLASS THAT DENIES ALL ACCESS TO ROWS AND COLUMNS TO A TABLE
204
ASSOCIATE AUTHORIZATION CLASS WITH TABLE TO ENFORCE A DEFAULT RULE OF "NO ACCESS" TO ROWS AND COLUMNS OF THE TABLE
206

FIG. 2

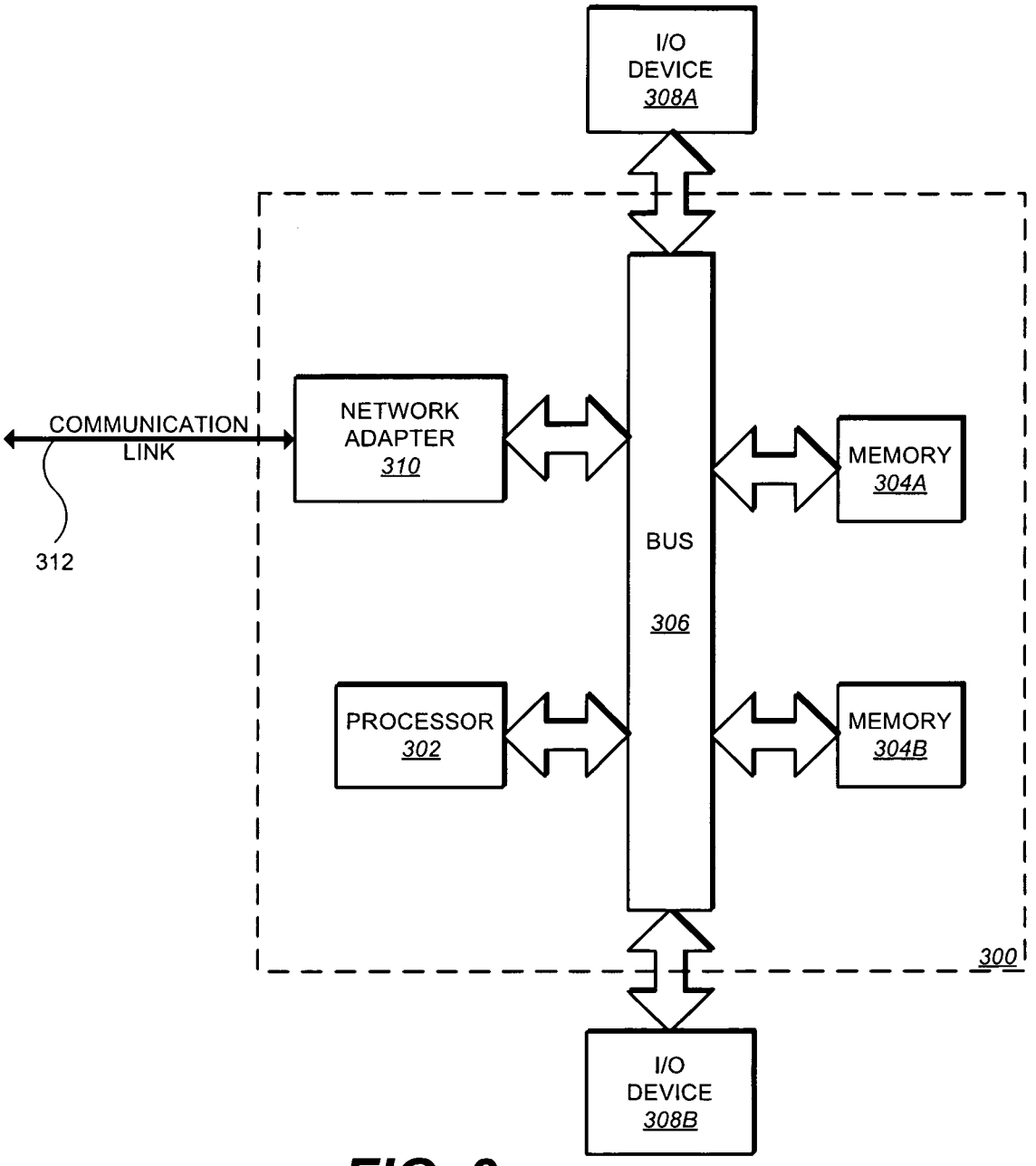

I/O
DEVICE
308A
COMMUNICATION
LINK
312
NETWORK
ADAPTER
310
BUS
306
MEMORY
304A
PROCESSOR
302
MEMORY
304B
300
I/O
DEVICE
308B

FIG. 3

METHOD AND SYSTEM FOR USING FINE-GRAINED ACCESS CONTROL (FGAC) TO CONTROL ACCESS TO DATA IN A DATABASE

FIELD OF THE INVENTION

The present invention relates generally to data processing, and more particularly to techniques for controlling access to data in a database.

BACKGROUND OF THE INVENTION

Business enterprises typically maintain data in database. For both legal and business reasons, business enterprises are increasingly becoming sensitive to unauthorized access to data in their databases. One type database system that is commonly used by enterprise businesses is a relational database in which data is organized in rows and columns of one or more tables (or table objects). Accordingly, business enterprises are exploring and implementing a number of mechanisms to prevent inadvertent or unauthorized access to row and/or column data. In a relational database management system (RDBMS), table object privileges granted to a user control whether or not access to the data in the table object is allowed. In general, such privilege control does not conventionally extend to the column-level or the row-level.

One technique for controlling access to data in a table on a column-level or a row-level includes use of a label-based access control (LBAC) mechanism—i.e., unless a label of a user is compatible with a label associated with a row or column of a table, then the data for that row or column is not returned to the user. Business enterprises, however, have generally been less accepting of label-based access control mechanisms due to the restrictive nature of label components, the need to provide a label for rows and columns, the lack of flexibility in terms of what can be expressed within labels.

Business enterprises have turned to more flexible mechanisms—e.g., fine-grained access control (FGAC) mechanisms including views, triggers, Oracle's virtual private database, and so on. Such fine-grained access control mechanisms all have one thing in common—the mechanisms supplement, but do not supplant, access control provided by privileges. That is, if a user has a SELECT privilege on a table, the user has access to all row and column data in the table; with conventional fine-grained access control mechanisms, that access is restricted by the addition of predicates and other logic to reduce the rows (and columns) seen by the user. But, by default, every user with privileges on a table has full access to all row and column data until and unless a fine-grained access control restriction is applied to rows or columns. This leaves open the possibility that a user, with privileges on a table object, can inadvertently be missed or not affected by fine-grained access control mechanisms, and therefore the user may be able to access data that the user would otherwise not be allowed to access.

BRIEF SUMMARY OF THE INVENTION

In general, this specification describes a method, system, and computer program for method for controlling access to data stored in a table of a database. In one implementation, the method includes marking the table of the database as being protected with fine-grained access control (FGAC), creating a system authorization class for the table of the database, the system authorization class having a default row authorization that prevents access to all rows in the table, the system authorization class being unmodifiable, creating a user authorization class for the table of the database, the user authorization class having a default row authorization that prevents access to all rows in the table, the user authorization class being modifiable, and associating the system authorization class and the user authorization class with the table of the database.

The details of one or more implementations are set forth in the accompanying drawings and the description below. Other features and advantages will be apparent from the description and drawings, and from the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a data processing system including a data access control module in accordance with one implementation.

FIG. **2** illustrates one implementation of a method for controlling access to data in a table of a database.

FIG. **3** is a block diagram of a data processing system suitable for assisting a user in creating software code in accordance with one implementation.

Like reference symbols in the various drawings indicate like elements.

DETAILED DESCRIPTION OF THE INVENTION

The present invention relates generally to data processing, and more particularly to techniques for controlling access to data in a database system. The following description is presented to enable one of ordinary skill in the art to make and use the invention and is provided in the context of a patent application and its requirements. The present invention is not intended to be limited to the implementations shown but is to be accorded the widest scope consistent with the principles and features described herein.

FIG. **1** illustrates a data processing system **100** in accordance with one implementation. Data processing system **100** includes input and output devices **102**, a programmed computer **104**, and a database **106**. Input and output devices **102** can include devices such as a printer, a keyboard, a mouse, a digitizing pen, a display, a printer, and the like. Programmed computer **104** can be any type of computer system, including for example, a workstation, a desktop computer, a laptop computer, a personal digital assistant (PDA), a cell phone, a network, and so on. Database **106** can be a relational database including one or more tables (not shown) for storing data.

Running on programmed computer **104** is a database management system (DBMS) **108** including a data access control module **110**. In one implementation, the database management system (DBMS) **108** and data access control module **110** are features of DB2 available from International Business Machines, Corporation of Armonk, N.Y. In one implementation, the data access control module **110** implements a fine-grained access control (FGAC) to control user access to data stored in one or more tables of the database **106**. The FGAC can be used to deny access to particular row(s) or column(s) of the one or more tables in the database **106**, which will be discussed in greater detail below.

In one implementation, the data access control module **110** implements row authorization and column authorization as the FGAC on the one or more tables in database **106**. In addition to the row/column authorization, there may also be traditional object-level (or table-level) privileges on each table (e.g., SELECT privilege on a table EMPLOYEE). In one implementation, a row authorization allows the holder of such authorization access to a subset of rows of an FGAC protected table. In one implementation, a column authoriza-

tion allows the holder of such authorization access to a subset of values (or cells) in a column of an FGAC protected table. In one implementation, row authorizations take precedence over column authorizations—i.e., if a user is not authorized to see any rows in an FGAC protected table, a column authorization for some column in that table will not allow that user to see any values in that column.

In one implementation, row and column authorizations are associated with a higher level entity called an authorization class. An authorization class (in one implementation) is associated with one and only one FGAC protected table, and contains one or more row authorizations and zero or more column authorizations. When an authorization class is created, a default row authorization that denies all access (e.g., a row predicate of "1=0") is created and implicitly granted to PUBLIC. This default row authorization cannot be deleted from the class nor can the default row authorization be revoked as it represents the default access available to users (which is none) through this authorization class.

FIG. **2** illustrates a method **200** for controlling access to data in a table of a database in accordance with one implementation. A table (e.g., in database **106**) is marked as being protected on a table-level (step **202**). For example, the table can be marked as being protected on a table-level by a database management system assigning one or access privileges to the table—i.e., access to the table is defined to the table as a whole. In one implementation, a fine-grained access control mechanism is applied to the table to protect (or control access to) the table on a table-level. Other suitable techniques for protecting access to a table on a table-level (or object-level) can be implemented. An authorization class is created (e.g., by the database management system) that, as a default, prevents access to all rows and columns of a table (step **204**). In one implementation, a default system authorization class is created as well as a default user authorization class. Both of these classes contain the normal class default row authorization described above. The system authorization class enforces the default access rule of "no access" for an FGAC protected table and this class cannot be dropped or modified in any way. The default user authorization class is provided as a location for any authorizations for which no authorization class is specified (i.e. it is a convenience to allow for authorizations to be defined without creating an authorization class); while the default user authorization class cannot be dropped, it can be modified like any other user defined authorization class.

An authorization class can be granted to (or revoked from) users, roles, groups, or PUBLIC. Granting an authorization class implicitly includes all authorizations defined within that class. If subsequent changes are made to the contents of that authorization class, those changes are automatically inherited by anyone granted the authorization class. If desired, row authorizations and column authorizations from an authorization class can be individually granted (with the exception of the class default row authorization); this would be of value in those cases where one or more of the authorizations, but not all, within an authorization class are to be granted or where there is no desire to have future changes to the contents of the authorization class automatically inherited.

The authorization class (or classes) is associated with the table to enforce a default rule of "no access" to rows and columns of the table (step **206**). All authorization classes defined on the same table affect and are considered for each and every query against that table. When more than one authorization of the same type (e.g., row or column) from the same authorization class apply to the same user, these authorizations are logically OR'ed together allowing that user access to the union of data authorized through those authorizations. For example, if user Joe is authorized to see all blue rows according to one row authorization in class AC**1**, and is authorized to see all red rows according to another authorization from the same authorization class AC**1**, then user Joe is allowed access to the union of blue and red rows.

By default, (in one implementation) the contents of different authorization classes on the same table are logically OR'ed together to achieve a union. However, sometimes this is not the desired behavior—i.e., in some cases, the contents of one or more of the authorization classes are considered to refine the contents of other authorization classes and the desire is to have the intersection of these authorization classes be used rather than the union. In such cases, the relationship between two authorization classes can be explicitly defined to be an intersect and the aggregate of authorizations present in the query from each authorization class will be logically AND'ed together instead. Specifically, when two classes are defined as intersecting, authorizations from the same authorization class will be OR'ed together to form a set and then logically AND'ed with the set from the other authorization class. For example, if user Joe is authorized to see all blue rows according to a row authorization in one authorization class AC**1**, and is authorized to see all rows for Canadian residents according to another authorization from a different authorization class AC**2**, where authorization class AC**2** has been defined as intersecting with authorization class AC**1**, then user Joe is authorized to see a view that contains the blue rows representing Canadian residents only (not all the blue rows). An authorization class can be defined to intersect with one or more (or all) authorization classes on the same FGAC protected table.

EXAMPLE

The following example illustrates one implementation of the techniques discussed above. Assume the following environment:

```
CREATE TABLE MYSCHEMA.T1 (C1 INT, C2 INT
  WITH ALTERNATE VALUE 99, C3 INT) PRO-
  TECTED BY FGAC
CREATE ROLE WAREHOUSE
CREATE ROLE ACCOUNTING
CREATE ROLE TEMPORARY_ACCOUNTING
GRANT SELECT ON MYSCHEMA.T1 TO ROLE
  WAREHOUSE, ROLE ACCOUNTING, ROLE TEM-
  PORARY_ACCOUNTING
GRANT ROLE WAREHOUSE TO FERNANDO
GRANT ROLE ACCOUNTING TO BOB
GRANT ROLE TEMPORARY_ACCOUNTING TO
  HALEY
```

The introduction of FGAC protection causes two authorization classes to be created:

- the system authorization class SYSIBM_DEFAULT containing the row authorization ROWDEFAULT with the (1=0) predicate which is implicitly granted to PUBLIC
- the default user authorization class USER_DEFAULT containing the row authorization ROWDEFAULT with the (1=0) predicate which is implicitly granted to PUBLIC

Bob, Fernando, and Haley all have SELECT privilege on MYSCHEMA.T1 from their role membership, but they do not have access to any rows in that table. If any of them issues a SELECT * FROM MYSCHEMA.T1, the internal representation of the query within the SQL compiler is the equivalent of:

```
SELECT * FROM (SELECT C1, 99, C3 FROM
  MYSCHEMA.T1 WHERE (1=0) OR (1=0))
```

Observe that there are two "1=0" predicates injected in SELECT query statement. The first predicate represents the default row authorization contained by the default user authorization class that was created when the table was marked as FGAC protected; the second predicate represents the default row authorization from the system defined authorization class. Since there are no column authorizations granted to any of them, DB2 injects just the alternate value for column C2 in the column. (NOTE: the SQL compiler is smart enough to remove the redundant 1=0 predicates above but leaving them in makes the description easier to follow).

Let us assume that the job definition for the members of the ACCOUNTING role requires them to see all rows where the column C1 equals 5. To allow this, a row authorization needs to be created and granted to the role. The security administrator decides to create an authorization class to represent the access needed for the ACCOUNTING job definition, creates a row authorization within the authorization class, and grants the set as a whole to the ACCOUNTING role.

```
CREATE AUTHORIZATION CLASS ACCOUNTING
  ON MYSCHEMA.T1
CREATE AUTHORIZATION ROWAUTH1 WITHIN
  MYSCHEMA.T1 ACCOUNTING
  FOR ROWS WHERE C1=5
GRANT AUTHORIZATION CLASS
  MYSCHEMA.T1.ACCOUNTING TO ROLE
  ACCOUNTING
```

Now, if Bob issues a SELECT * FROM MYSCHEMA.T1, he will be able to access some rows in this table based on the following reasons. First, Bob has SELECT privilege on MYSCHEMA.T1 granted to him via the role ACCOUNTING. Second, this same role has been granted an authorization class, ACCOUNTING, defined on table MYSCHEMA.T1. The ACCOUNTING class contains a row authorization which allows Bob to see all rows in MYSCHEMA.T1 where column C1=5. However, Bob does not hold (directly or indirectly) a column authorization for protected column C2. Therefore, Bob will still see the alternate value 99 for all rows in MYSCHEMA.T1 where column C1=5. The internal representation of the query within the SQL compiler is the equivalent of:

```
SELECT * FROM (SELECT C1, 99, C3 FROM
  MYSCHEMA.T1
  WHERE ((C1=5) OR (1=0)) OR (1=0) OR (1=0))
```

The first row predicate of ((C1=5) OR (1=0)) represents the authorizations granted to Bob indirectly when the authorization class ACCOUNTING was granted to the role ACCOUNTING. The first row predicate represents all the current authorizations in this authorization class. The next row predicate (1=0) is the default row authorization from the default user authorization class. The last row predicate (1=0) is the default row authorization from the system defined authorization class. Since no class intersects with any other, the predicates are OR'ed together to get the union.

To allow Bob access to values in column C2, a column authorization must be defined and granted to him, or to a role he is member in, or to a group he is member in, or to PUBLIC. Let's assume that the ACCOUNTING job definition requires access to column C2 so the security administrator defines a new column authorization in the existing ACCOUNTING authorization class which contains a condition allowing access only to a set of specific values in column C2.

```
CREATE AUTHORIZATION COLUMNAUTH1
  WITHIN ACCOUNTING
  FOR COLUMN C2 WHERE C2>10
```

Now, if Bob issues a SELECT * FROM MYSCHEMA.T1, the internal representation of the query within the SQL compiler is the equivalent of:

```
SELECT * FROM (SELECT C1, (CASE WHEN C2>10
  THEN C2 ELSE 99 END), C3 FROM MYSCHE-
  MA.T1
  WHERE ((C1=5) OR (1=0)) OR (1=0) OR (1=0))
```

The row predicates are as they were in the previous case but now Bob has automatically inherited the new column authorization in the ACCOUNTING authorization class as well.

Meanwhile, Haley is still unable to access any rows in the table. As a temporary employee in accounting, let us assume that she is only allowed to see the same rows as Bob but not the contents of column C2. The security administrator could define an authorization class to represent this particular case, but instead the security administrator chooses to simply grant the ROWAUTH1 authorization, but not the ACCOUNTING authorization class itself, directly to the role TEMPORARY_ACCOUNTING since the security administrator plans to later remove it (i.e., it is a temporary solution:)

```
GRANT AUTHORIZATION ROWAUTH1 WITHIN
  MYSCHEMA.T1.ACCOUNTING TO ROLE TEMPO-
  RARY_ACCOUNTING
```

Now, if Haley issues a SELECT * FROM MYSCHEMA.T1, the internal representation of the query within the SQL compiler is the equivalent of:

```
SELECT * FROM (SELECT C1, 99, C3 FROM
  MYSCHEMA.T1
  WHERE ((C1=5) OR (1=0)) OR (1=0) OR (1=0))
```

Since Haley does not have column authorization for column C2, she will simply get the alternate value. Also, since she was granted a specific authorization and not the authorization class, she will not automatically inherit the rest of the authorizations, or any future changes, that exist in the class.

Suppose that the security administrator wishes to stop all access as he tracks a security problem. To do so, the security administrator quickly alters the default user authorization class, which currently only has the default row authorization, to intersect with all other authorization classes on the table as follows:

```
ALTER AUTHORIZATION CLASS USER_DEFAULT
  ON MYSCHEMA.T1 INTERSECTS WITH ALL
```

At this point, if Bob or Haley issues a SELECT * FROM MYSCHEMA.T1, they will see no rows at all for the following reason. The change to make the USER_DEFAULT authorization class intersect with all other authorization classes now means that the granted authorizations from USER_DEFAULT, in this case the default row authorization for the class, are logically AND'ed with all the others. The internal representation of the query within the SQL compiler is the equivalent of:

```
SELECT * FROM (SELECT C1, (CASE WHEN C2>10
  THEN C2 ELSE 99 END), C3 FROM MYSCHE-
  MA.T1
  WHERE ((1=0) AND (((C1=5) OR (1=0)) OR (1=0)))
```

In this case, the relevant authorizations from the intersecting authorization class have been placed in the first predicate and then logically AND'ed with the union of the relevant authorizations from all the other classes. Obviously, Bob sees no rows this way. To remove the emergency access stoppage, the security administrator modifies the USER_DEFAULT authorization class so that is not longer intersecting with all others. Accordingly, authorization classes can dynamically adjust to change (e.g., changes to class are automatically seen by all who have access to class).

Fernando can still not see any rows as nothing has changed for him. As a member of the Warehouse team, it is decided

that Fernando is allowed to see any rows where column C3<100. The security administrator decides not to create a new authorization class for this case and does the following:

```
CREATE AUTHORIZATION ROWAUTH2
  FOR ROWS WHERE (C3<100)
```

This causes a row authorization to be created in the USER_DEFAULT authorization class. The security administrator now grants this to the Warehouse role so that Fernando acquires the row authorization, as follows:

```
GRANT AUTHORIZATION ROWAUTH2 WITHIN
  MYSCHEMA.T1.USER_DEFAULT TO ROLE
  WAREHOUSE
```

Now, if Fernando issues a SELECT * FROM MYSCHEMA.T1, the internal representation of the query within the SQL compiler is the equivalent of:

```
SELECT * FROM (SELECT C1, 99, C3 FROM
  MYSCHEMA.T1
  WHERE ((1=0)) OR ((C3<100) OR (1=0)) OR (1=0))
```

Since Fernando does not have column authorization for column C2, he will simply get the alternate value. The first row predicate (1=0) is the default row predicate from the ACCOUNTING authorization class while the second row predicate ((C3<100) OR (1=0)) shows the union of all authorizations available to Fernando in the USER_DEFAULT authorization class. The last row predicate is the default row predicate from the system defined authorization class.

Suppose that the security administrator wishes to limit the rows that can be seen on the weekend by anyone in accounting to those for which column C3 is equal to zero. To do so, the security administrator creates a new authorization class WEEKEND_ACCESS that intersects with authorization class ACCOUNTING as follows:

```
CREATE AUTHORIZATION CLASS WEEKEND_AC-
  CESS ON MYSCHEMA.T1 Intersects with Accounting
CREATE AUTHORIZATION ROWAUTH3 WITHIN
  WEEKEND_ACCESS
  FOR ROWS WHERE (IS_WEEKEND( ) AND C3=0)
GRANT AUTHORIZATION CLASS MYSCHEMA.T1
  WEEKEND_ACCESS TO PUBLIC
```

Now, if Bob issues a SELECT * FROM MYSCHEMA.T1, the internal representation of the query within the SQL compiler is the equivalent of:

```
SELECT * FROM (SELECT C1, (CASE WHEN C2>10
  THEN C2 ELSE 99 END), C3 FROM MYSCHE-
  MA.T1
  WHERE (((IS_WEEKEND( ) AND C3=0) OR (1=0))
    AND ((C1=5) OR (1=0))) OR (1=0) OR (1=0)))
```

In this case, the predicate ((IS_WEEKEND( ) AND C3=0) OR (1=0)) represents all the relevant authorizations from the new authorization class WEEKEND_ACCESS and these are logically AND'ed with all the relevant authorizations from the intersecting authorization class ACCOUNTING in the form of the predicate ((C1=5) OR (1=0)). Finally, the relevant authorizations from the other, non-intersecting authorization classes are OR'ed in (for this example, they are simply the class default authorizations for the system and user default classes).

Note that this new intersecting class also affects Haley but not Fernando. If Haley issues a SELECT * FROM MYSCHEMA.T1, the internal representation of the query within the SQL compiler is the equivalent of:

```
SELECT * FROM (SELECT C1, 99, C3 FROM
  MYSCHEMA.T1
  WHERE (((IS_WEEKEND( ) AND C3=0) OR (1=0))
    AND ((C1=5) OR (1=0))) OR (1=0) OR (1=0)))
```

Since Haley's access is dependent on the authorizations in the ACCOUNTING class, the new authorization WEEKEND_ACCESS class can close off that access since its authorizations are AND'ed with those in the ACCOUNTING class. If Fernando issues a SELECT * FROM MYSCHEMA.T1, the internal representation of the query within the SQL compiler is the equivalent of:

```
SELECT * FROM (SELECT C1, 99, C3 FROM
  MYSCHEMA.T1
  WHERE (((IS_WEEKEND( ) AND C3=0) OR (1=0))
    AND ((1=0))) OR ((C3<100) OR (1=0)) OR (1=0)))
```

In this case, Fernando is not dependent on authorizations from ACCOUNTING and so his access is not affected by the new authorization class.

In the example above, rather than modify the column definition to implement FGAC, an administrator can simply create a column authorization as follows:

```
CREATE AUTHORIZATION AUTHx
ON TABLE T1
FOR COLUMN C2
(Case when C2>10 then C2 Else 99 End)
```

Hence, the alternate value need not be specified together with the table definition and could be done separately within the column authorization definition.

Implementation

In one implementation, SQL DDL statements are used to create authorization classes and authorizations as well as to grant and revoke the authorization classes. Modified SQL statements can be used to modify table attributes to activate FGAC protection. When an SQL/XML statement is compiled, for each reference to a table marked as FGAC protected, the authorization classes defined for that table, be it the one created explicitly by the administrator or the default one created by the system when the table is marked protected, are searched and any relevant row or column authorizations in that class for the statement authorization information (primary and secondary authorization IDs) are gathered; relevancy is determined by whether the authorization class, or individual authorization, has been granted to one of the authorization IDs in the statement authorization information.

A "pseudo-view" definition is created by: gathering all the relevant row authorizations from the same authorization class and logically OR'ing them together in a "authorization class expression"; identifying which authorization classes, if any, are defined as intersecting with each other and logically AND'ing the "authorization class expression" for each of these classes with the other to create a "intersecting authorization class expression" set; logically OR'ing any remaining "authorization class expression" with each other and then logically OR'ing them with all "intersecting authorization class expression" sets; using the final result as the predicate portion of the "pseudo-view" definition. Similar logic is followed for dealing with the expressions from all relevant column authorizations with the end result for each unique column being implemented as CASE logic in the appropriate location for the column in select list of the "pseudo-view" definition. If no column authorizations are found, then the defined alternate value is implemented as a constant in that location.

One or more of method steps described above can be performed by one or more programmable processors executing a computer program to perform functions by operating on input data and generating output. Generally, the invention can take the form of an entirely hardware embodiment, an entirely software embodiment or an embodiment containing both hardware and software elements. In one implementation, the invention is implemented in software, which includes but is not limited to firmware, resident software, microcode, etc.

Furthermore, the invention can take the form of a computer program product accessible from a computer-usable or computer-readable medium providing program code for use by or in connection with a computer or any instruction execution system. For the purposes of this description, a computer-usable or computer readable medium can be any apparatus that can contain, store, communicate, propagate, or transport the program for use by or in connection with the instruction execution system, apparatus, or device. The medium can be an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system (or apparatus or device) or a propagation medium. Examples of a computer-readable medium include a semiconductor or solid state memory, magnetic tape, a removable computer diskette, a random access memory (RAM), a read-only memory (ROM), a rigid magnetic disk and an optical disk. Current examples of optical disks include compact disk—read only memory (CD-ROM), compact disk—read/write (CD-R/W) and DVD.

FIG. **3** illustrates a data processing system **300** suitable for storing and/or executing program code. Data processing system **300** includes a processor **302** coupled to memory elements **304**A-B through a system bus **306**. In other implementations, data processing system **300** may include more than one processor and each processor may be coupled directly or indirectly to one or more memory elements through a system bus. Memory elements **304**A-B can include local memory employed during actual execution of the program code, bulk storage, and cache memories that provide temporary storage of at least some program code in order to reduce the number of times the code must be retrieved from bulk storage during execution. As shown, input/output or I/O devices **308**A-B (including, but not limited to, keyboards, displays, pointing devices, etc.) are coupled to data processing system **300**. I/O devices **308**A-B may be coupled to data processing system **300** directly or indirectly through intervening I/O controllers (not shown).

In one implementation, a network adapter **310** is coupled to data processing system **300** to enable data processing system **300** to become coupled to other data processing systems or remote printers or storage devices through communication link **312**. Communication link **312** can be a private or public network. Modems, cable modems, and Ethernet cards are just a few of the currently available types of network adapters.

Various implementations for controlling access to data in a database have been described. Nevertheless, various modifications may be made to the implementations. For example, steps of the methods described above can be performed in a different order and still achieve desirable results. Accordingly, many modifications may be made without departing from the scope of the following claims.

What is claimed is:

1. A computer-implemented method for controlling access to data stored in a table of a database, wherein the computer performs the functions in the following method, the method comprising:

marking the table of the database as being protected with fine-grained access control (FGAC);

creating a system authorization class for the table of the database, the system authorization class having a default row-level authorization that prevents access to all rows in the table, the system authorization class being unmodifiable;

creating a user authorization class for the table of the database, the user authorization class having a default row-level authorization that prevents access to all rows in the table, the user authorization class being modifiable wherein the user authorization class is provided as a location for any authorizations for which no authorization class is specified;

associating the system authorization class and the user authorization class with the table of the database, wherein the association of the system authorization class with the table of the database operates to deny access to the rows and columns of the table;

receiving a request from a user seeking to access data in the table of the database;

determining whether any other user authorization class is applicable to the user;

responsive to no other user authorization class being applicable to the user,

determining whether the system authorization class and the user authorization class are defined as intersecting classes;

responsive to the system authorization class and the user authorization class being defined as intersecting classes, preventing the user from accessing any row in the table of the database;

responsive to the system authorization class and the user authorization class not being defined as intersecting classes, permitting the user to access rows or columns in the table of the database based on the union of authorizations.

2. The computer-implemented method of claim **1** further comprising:

responsive to the system authorization class and the user authorization class not being defined as intersecting classes,

forming a union of authorizations by logically OR'ing authorizations from the system authorization class and the user authorization class.

3. The computer-implemented method of claim **2**, wherein responsive to at least one other user authorization class being applicable to the user, the method further comprises:

determining whether the system authorization class and the at least one other user authorization class are defined as intersecting classes; and

responsive to the system authorization class and the at least one other user authorization class being defined as intersecting classes, preventing the user from accessing any row in the table of the database.

4. The computer-implemented method of claim **3**, wherein responsive to the system authorization class and the at least one other user authorization class not being defined as intersecting classes, the method further comprises:

determining whether the user authorization class and the at least one other user authorization class are defined as intersecting classes;

responsive to the user authorization class and the at least one other user authorization class being defined as intersecting classes,

forming a first set of authorizations by logically OR'ing authorizations from the user authorization class;

forming a second set of authorizations by logically OR'ing authorizations from the at least one other user authorization class;

forming an intersection of authorizations by logically AND'ing the first set of authorizations and the second set of authorizations; and

permitting the user to access rows or columns in the table of the database based on the intersection of authorizations.

5. The computer-implemented method of claim **4**, wherein responsive to the user authorization class and the at least one

other user authorization class not being defined as intersecting classes, the method further comprises:

forming a union of authorizations by logically OR'ing authorizations from the user authorization class and the at least one other user authorization class; and

permitting the user to access rows or columns in the table of the database based on the union of authorizations.

6. The computer-implemented method of claim **1**, wherein the database is a relational database.

7. A non-transitory computer program product comprising a non-transitory computer readable storage medium, the non-transitory computer readable storage medium for controlling access to data stored in a table of a database, the computer program comprising computer executable code for:

marking the table of the database as being protected with fine-grained access control (FGAC);

creating a system authorization class for the table of the database, the system authorization class having a default row-level authorization that prevents access to all rows in the table, the system authorization class being unmodifiable wherein the user authorization class is provided as a location for any authorizations for which no authorization class is specified;

creating a user authorization class for the table of the database, the user authorization class having a default row-level authorization that prevents access to all rows in the table, the user authorization class being modifiable; and

associating the system authorization class and the user authorization class with the table of the database, wherein the association of the system authorization class with the table of the database operates to deny access to the rows and columns of the table;

receiving a request from a user seeking to access data in the table of the database;

determining whether any other user authorization class is applicable to the user;

responsive to no other user authorization class being applicable to the user,

determining whether the system authorization class and the user authorization class are defined as intersecting classes;

responsive to the system authorization class and the user authorization class being defined as intersecting classes, preventing the user from accessing any row in the table of the database;

responsive to the system authorization class and the user authorization class not being defined as intersecting classes, permitting the user to access rows or columns in the table of the database based on the union of authorizations.

8. The non-transitory computer product of claim **7**, wherein the computer program product further comprises computer executable code for:

responsive to the system authorization class and the user authorization class not being defined as intersecting classes,

forming a union of authorizations by logically OR'ing authorizations from the system authorization class and the user authorization class.

9. The non-transitory computer program product of claim **8**, wherein responsive to at least one other user authorization class being applicable to the user, the computer program product further comprises computer executable code for:

determining whether the system authorization class and the at least one other user authorization class are defined as intersecting classes; and

responsive to the system authorization class and the at least one other user authorization class being defined as intersecting classes, preventing the user from accessing any row in the table of the database.

10. The non-transitory computer program product of claim **9**, wherein responsive to the system authorization class and the at least one other user authorization class not being defined as intersecting classes, the computer program product further comprises computer executable code for:

determining whether the user authorization class and the at least one other user authorization class are defined as intersecting classes;

responsive to the user authorization class and the at least one other user authorization class being defined as intersecting classes,

forming a first set of authorizations by logically OR'ing authorizations from the user authorization class;

forming a second set of authorizations by logically OR'ing authorizations from the at least one other user authorization class;

forming an intersection of authorizations by logically AND'ing the first set of authorizations and the second set of authorizations; and

permitting the user to access rows or columns in the table of the database based on the intersection of authorizations.

11. The non-transitory computer program product of claim **10**, wherein responsive to the user authorization class and the at least one other user authorization class not being defined as intersecting classes, the computer program product further comprises computer executable code for:

forming a union of authorizations by logically OR'ing authorizations from the user authorization class and the at least one other user authorization class; and

permitting the user to access rows or columns in the table of the database based on the union of authorizations.

12. The non-transitory computer program product of claim **7**, wherein the database is a relational database.

13. A non-transitory computer system comprising:

a processing system;

a storage medium;

a database; and

a database management system controlling access to data stored in a table of the database, the database management system

marking the table of the database as being protected with fine-grained access control (FGAC);

creating a system authorization class for the table of the database, the system authorization class having a default row-level authorization that prevents access to all rows in the table, the system authorization class being unmodifiable;

creating a user authorization class for the table of the database, the user authorization class having a default row-level authorization that prevents access to all rows in the table, the user authorization class being modifiable wherein the user authorization class is provided as a location for any authorizations for which no authorization class is specified;

associating the system authorization class and the user authorization class with the table of the database, wherein the association of the system authorization class with the table of the database operates to deny access to the rows and columns of the table;

receiving a request from a user seeking to access data in the table of the database;

determining whether any other user authorization class is applicable to the user;

responsive to no other user authorization class being applicable to the user,

determining whether the system authorization class and the user authorization class are defined as intersecting classes;

responsive to the system authorization class and the user authorization class being defined as intersecting classes, preventing the user from accessing any row in the table of the database;

responsive to the system authorization class and the user authorization class not being defined as intersecting classes, permitting the user to access rows or columns in the table of the database based on the union of authorizations.

14. The non-transitory computer system of claim **13**, wherein the database management system further

responsive to the system authorization class and the user authorization class not being defined as intersecting classes,

forms a union of authorizations by logically OR'ing authorizations from the system authorization class and the user authorization class.

15. The non-transitory computer system of claim **14**, wherein responsive to at least one other user authorization class being applicable to the user, the database management system further

determines whether the system authorization class and the at least one other user authorization class are defined as intersecting classes, and

responsive to the system authorization class and the at least one other user authorization class being defined as intersecting classes, prevents the user from accessing any row in the table of the database.

16. The non-transitory computer system of claim **15**, wherein responsive to the system authorization class and the at least one other user authorization class not being defined as intersecting classes, the database management system further

determines whether the user authorization class and the at least one other user authorization class are defined as intersecting classes,

responsive to the user authorization class and the at least one other user authorization class being defined as intersecting classes,

forms a first set of authorizations by logically OR'ing authorizations from the user authorization class,

forms a second set of authorizations by logically OR'ing authorizations from the at least one other user authorization class,

forms an intersection of authorizations by logically AND'ing the first set of authorizations and the second set of authorizations, and

permits the user to access rows or columns in the table of the database based on the intersection of authorizations.

17. The non-transitory computer system of claim **16**, wherein responsive to the user authorization class and the at least one other user authorization class not being defined as intersecting classes, the database management system further

forms a union of authorizations by logically OR'ing authorizations from the user authorization class and the at least one other user authorization class, and

permits the user to access rows or columns in the table of the database based on the union of authorizations.

18. The non-transitory computer system of claim **13**, wherein the database is a relational database.

* * * * *

US007647626B2

(12) **United States Patent**
Bird et al.

(10) **Patent No.: US 7,647,626 B2**
(45) **Date of Patent: *Jan. 12, 2010**

(54) **METHOD FOR ESTABLISHING A TRUSTED RELATIONSHIP BETWEEN A DATA SERVER AND A MIDDLEWARE SERVER**

(75) Inventors: **Paul Miller Bird**, Markham (CA); **Curt Lee Cotner**, Gilroy, CA (US); **Walid Rjaibi**, Markham (CA); **Timothy Jon Vincent**, Toronto (CA)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 745 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **11/008,507**

(22) Filed: **Dec. 8, 2004**

(65) **Prior Publication Data**

US 2006/0123468 A1 Jun. 8, 2006

(51) **Int. Cl.**

G06F 7/04	(2006.01)
G06F 15/16	(2006.01)
G06F 17/30	(2006.01)
H04L 29/06	(2006.01)

(52) **U.S. Cl.** **726/5**; 726/2; 726/3

(58) **Field of Classification Search** 726/2, 726/3, 5

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,586,260	A	12/1996	Hu	395/200.2
5,598,536	A *	1/1997	Slaughter et al.	709/219
5,619,657	A *	4/1997	Sudama et al.	709/225
5,841,869	A	11/1998	Merkling et al.	
6,052,785	A	4/2000	Lin et al.	713/201
6,076,092	A	6/2000	Goldberg et al.	707/103
6,112,196	A	8/2000	Zimowski et al.	707/2
6,212,636	B1 *	4/2001	Boyle et al.	713/168
6,266,666	B1	7/2001	Ireland et al.	707/10
6,286,104	B1	9/2001	Buhle et al.	713/201
6,349,338	B1 *	2/2002	Seamons et al.	709/229
6,377,994	B1 *	4/2002	Ault et al.	709/229

(Continued)

OTHER PUBLICATIONS

Park, J. S. and Sandhu, R. 2000. Secure Cookies on the Web. IEEE Internet Computing 4, 4 (Jul. 2000), 36-44.*

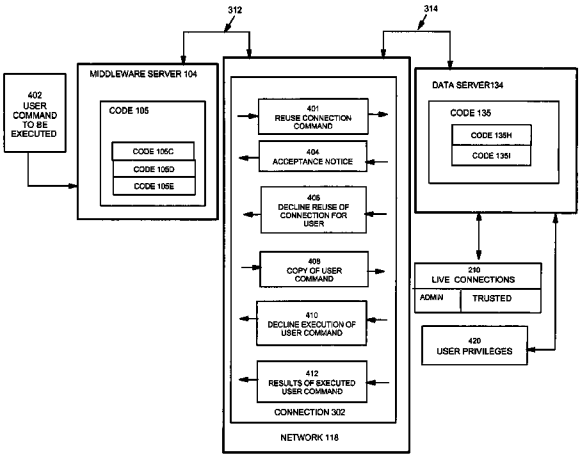
(Continued)

Primary Examiner—Christian LaForgia
Assistant Examiner—James Turchen
(74) *Attorney, Agent, or Firm*—Sughrue Mion, PLLC

(57) **ABSTRACT**

A data server of a data processing system is operably coupled to a database and in communication with a middleware server. A connection between the data server and the middleware server is established and managed. A set of attributes identifying trusted middleware servers is instituted with the data server. The middleware server transmits a connection request to the data server. The connection request has request attributes including identifying the connection request as being for a new connection or reuse of an existing connection with different connection request attributes. A connection with the middleware server is established by the data server based on the connection request. A connection status message is received by the middleware server from the data server indicating a status of the connection request. A trust indicator for the connection is established at the data server according to a trust status identified by the set of attributes for the middleware server.

36 Claims, 8 Drawing Sheets



U.S. PATENT DOCUMENTS

6,434,543	B1	8/2002	Goldberg et al.	707/2
6,516,416	B2 *	2/2003	Gregg et al.	726/8
6,631,371	B1	10/2003	Lei et al.	
6,745,332	B1 *	6/2004	Wong et al.	726/4
7,174,565	B2 *	2/2007	Kadyk et al.	726/12
7,181,764	B2 *	2/2007	Zhu et al.	726/4
7,325,246	B1 *	1/2008	Halasz et al.	726/2
2002/0016777	A1 *	2/2002	Seamons et al.	705/76
2002/0049914	A1	4/2002	Inoue et al.	713/201
2002/0065956	A1	5/2002	Yagawa et al.	709/330
2002/0184217	A1 *	12/2002	Bisbee et al.	707/9
2003/0014527	A1	1/2003	Terwindt et al.	709/227
2003/0236975	A1	12/2003	Birk et al.	
2004/0064335	A1 *	4/2004	Yang	705/1
2006/0075075	A1	4/2006	Malinen et al.	

OTHER PUBLICATIONS

Chadwick, D. W., Otenko, A., and Ball, E. 2003. Role-Based Access Control With X.509 Attribute Certificates. IEEE Internet Computing 7, 2 (Mar. 2003), 62-69.*

Kristol, D. and Montulli, L. 1997 HTTP State Management Mechanism. RFC. RFC Editor.*

* cited by examiner

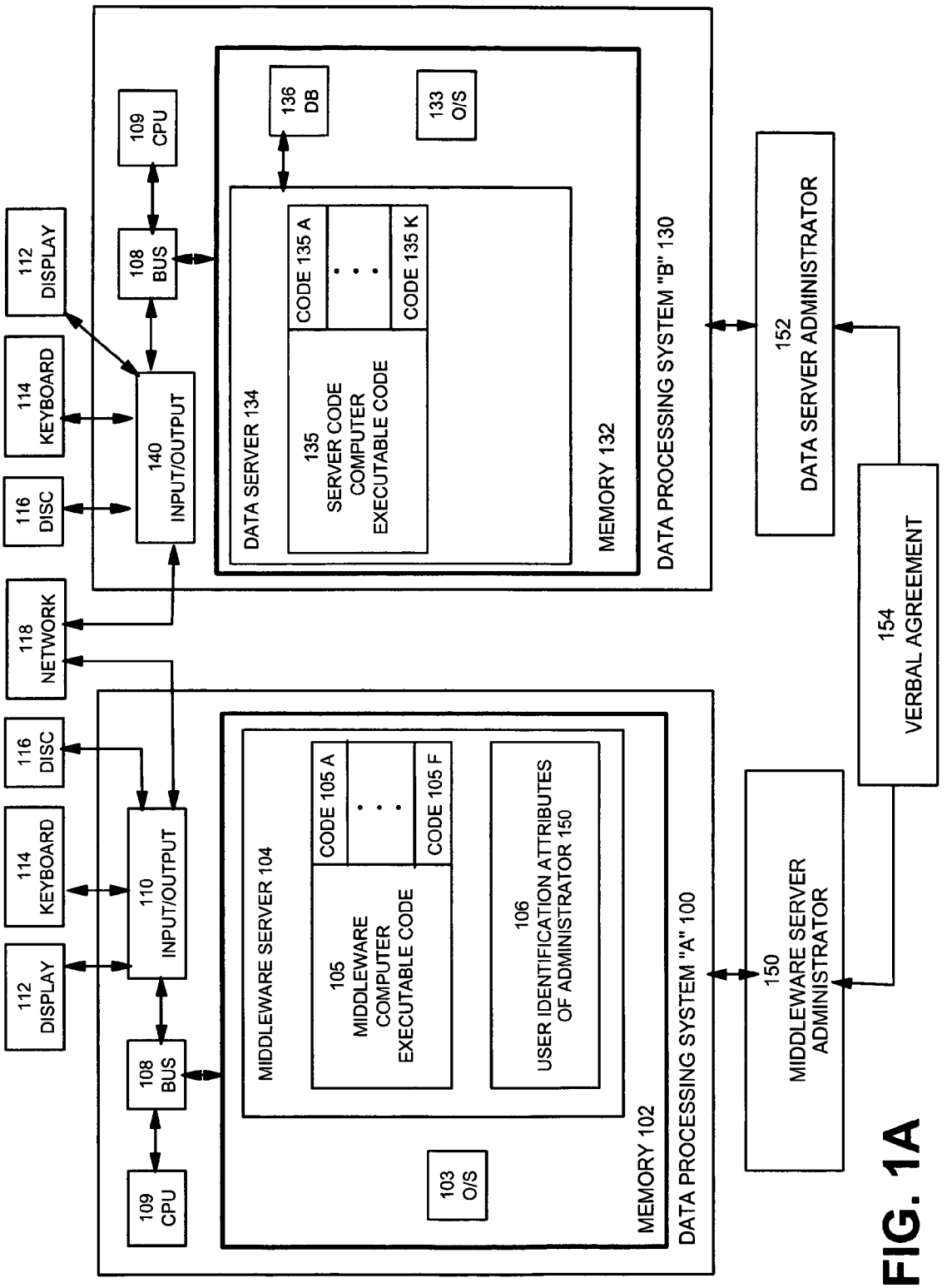

DATA PROCESSING SYSTEM "A" 100
MEMORY 102
MIDDLEWARE SERVER 104
105 MIDDLEWARE COMPUTER EXECUTABLE CODE
CODE 105 A
CODE 105 F
106 USER IDENTIFICATION ATTRIBUTES OF ADMINISTRATOR 150
103 O/S
109 CPU
108 BUS
110 INPUT/OUTPUT
112 DISPLAY
114 KEYBOARD
116 DISC
118 NETWORK
DATA PROCESSING SYSTEM "B" 130
MEMORY 132
DATA SERVER 134
135 SERVER CODE COMPUTER EXECUTABLE CODE
CODE 135 A
CODE 135 K
136 DB
133 O/S
109 CPU
108 BUS
140 INPUT/OUTPUT
112 DISPLAY
114 KEYBOARD
116 DISC
150 MIDDLEWARE SERVER ADMINISTRATOR
154 VERBAL AGREEMENT
152 DATA SERVER ADMINISTRATOR

FIG. 1A

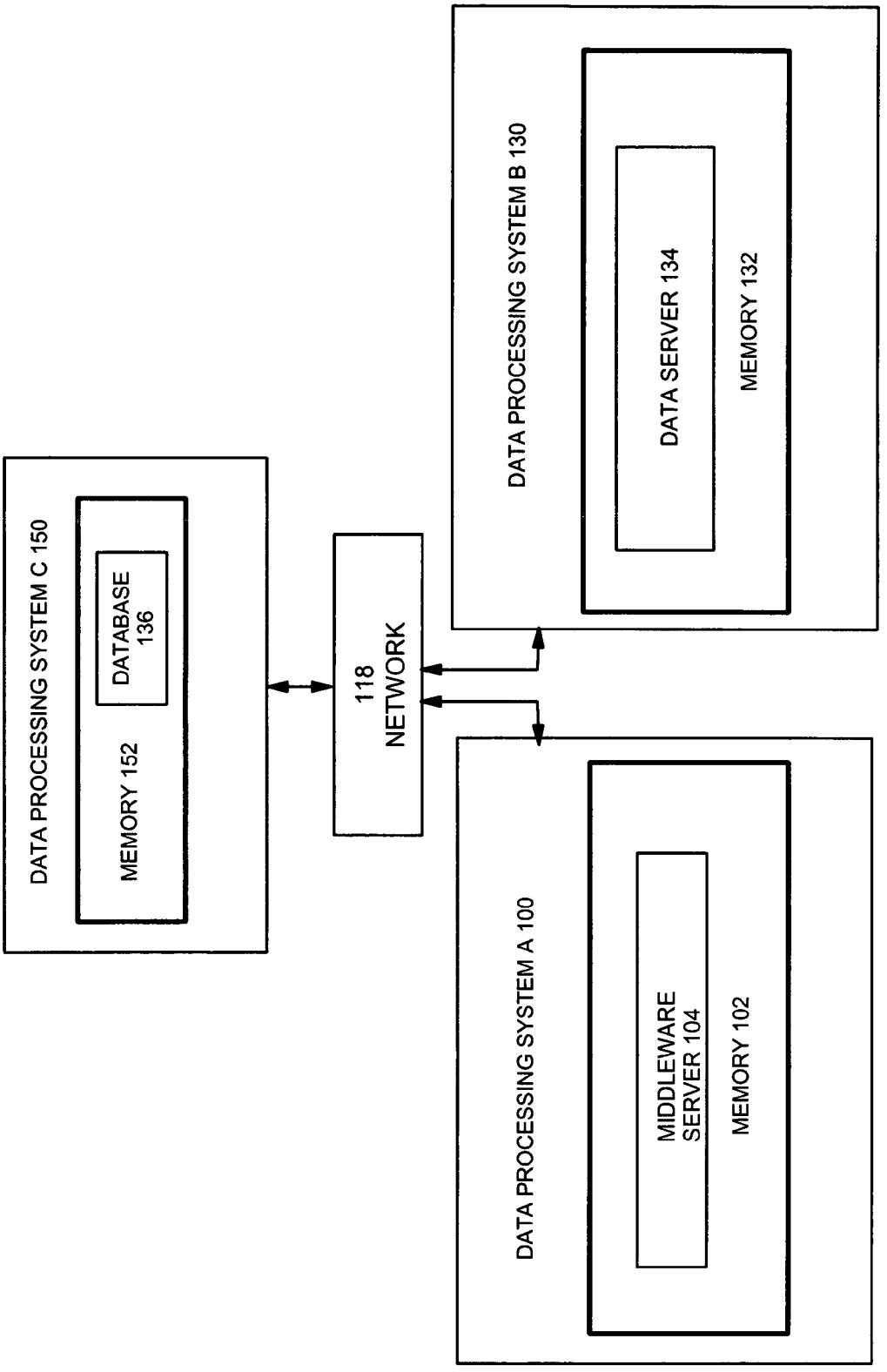

DATA PROCESSING SYSTEM C 150
MEMORY 152
DATABASE 136
118 NETWORK
DATA PROCESSING SYSTEM B 130
DATA SERVER 134
MEMORY 132
DATA PROCESSING SYSTEM A 100
MIDDLEWARE SERVER 104
MEMORY 102

FIG. 1B

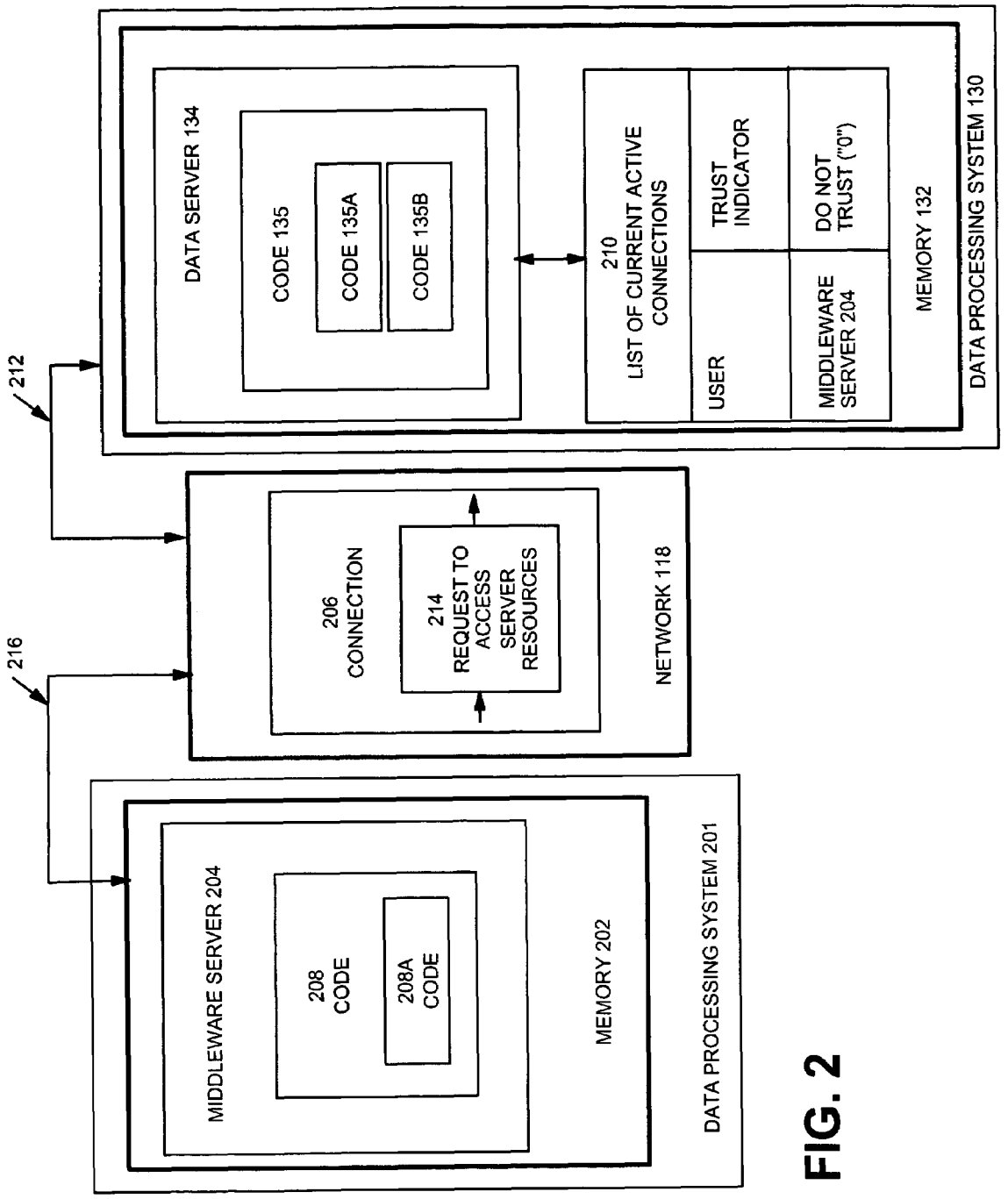

DATA PROCESSING SYSTEM 201
MEMORY 202
MIDDLEWARE SERVER 204
208 CODE
208A CODE
216
NETWORK 118
206 CONNECTION
214 REQUEST TO ACCESS SERVER RESOURCES
212
DATA PROCESSING SYSTEM 130
MEMORY 132
DATA SERVER 134
CODE 135
CODE 135A
CODE 135B
210 LIST OF CURRENT ACTIVE CONNECTIONS
USER
TRUST INDICATOR
MIDDLEWARE SERVER 204
DO NOT TRUST ("0")

FIG. 2

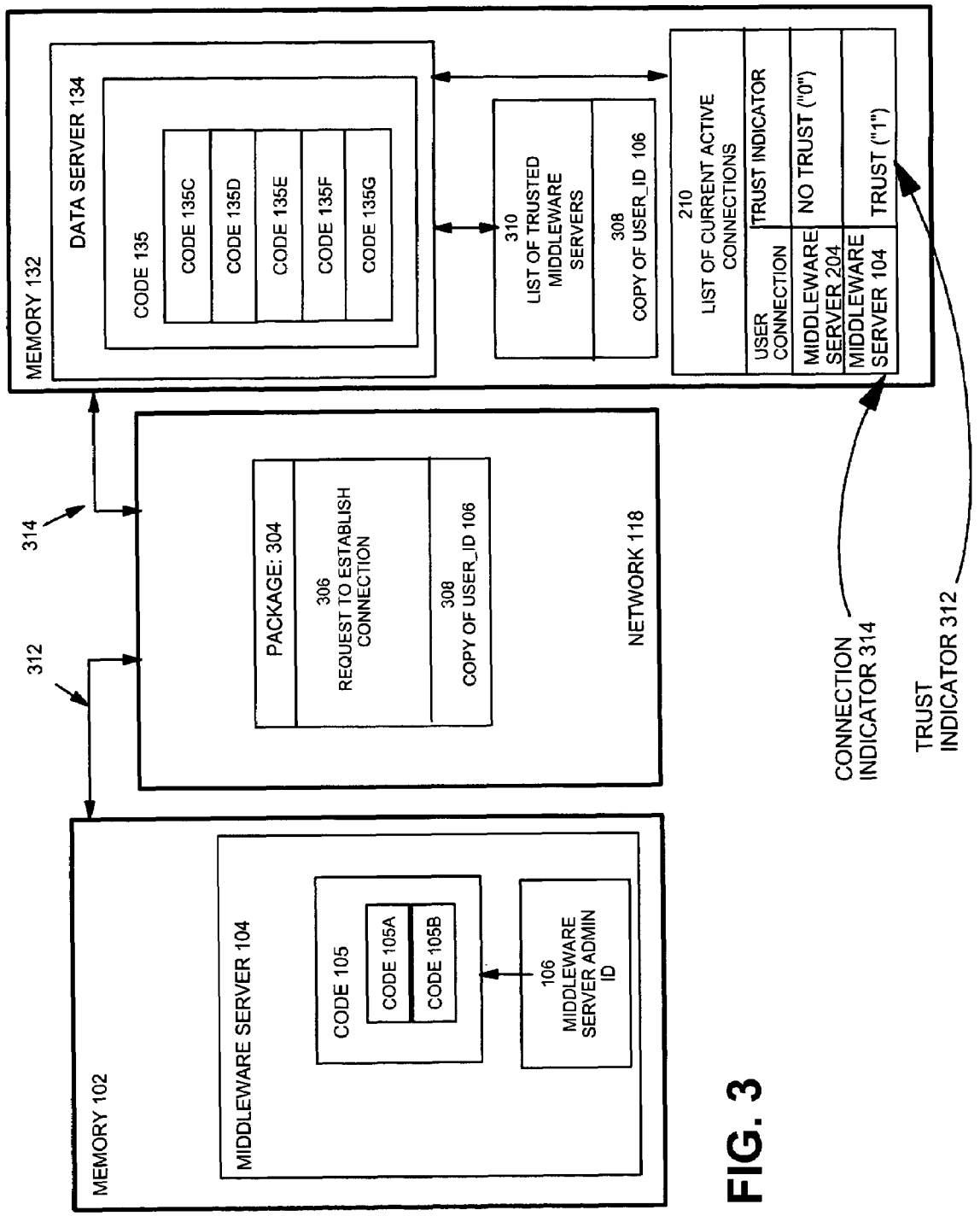

MEMORY 102
MIDDLEWARE SERVER 104
CODE 105
CODE 105A
CODE 105B
106
MIDDLEWARE SERVER ADMIN ID
312
314
NETWORK 118
PACKAGE: 304
306
REQUEST TO ESTABLISH CONNECTION
308
COPY OF USER_ID 106
MEMORY 132
DATA SERVER 134
CODE 135
CODE 135C
CODE 135D
CODE 135E
CODE 135F
CODE 135G
310
LIST OF TRUSTED MIDDLEWARE SERVERS
308
COPY OF USER_ID 106
210
LIST OF CURRENT ACTIVE CONNECTIONS
USER CONNECTION
TRUST INDICATOR
MIDDLEWARE SERVER 204
NO TRUST ("0")
MIDDLEWARE SERVER 104
TRUST ("1")
CONNECTION INDICATOR 314
TRUST INDICATOR 312

FIG. 3

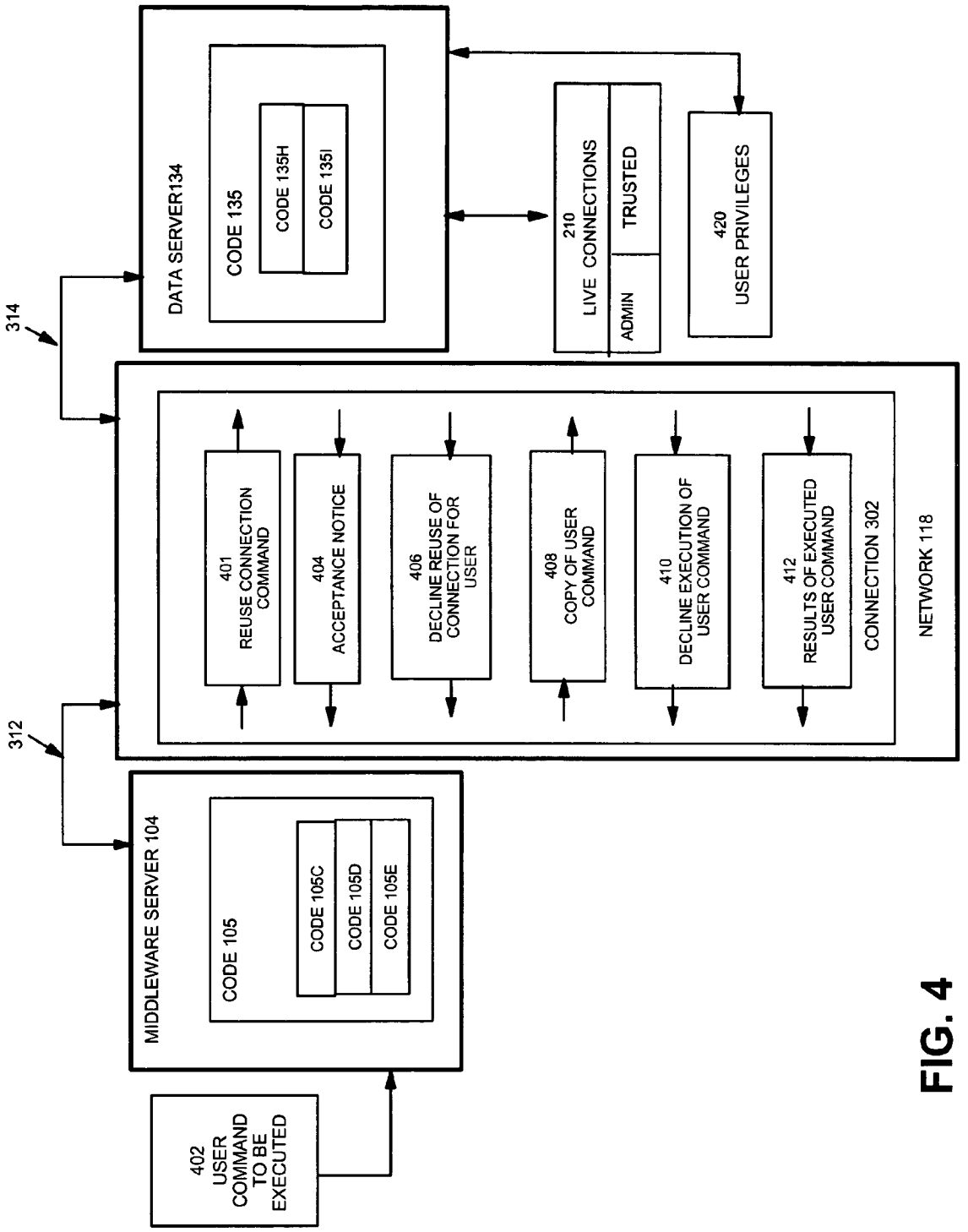

DATA SERVER134
CODE 135
CODE 135H
CODE 135I
210
LIVE CONNECTIONS
TRUSTED
ADMIN
420
USER PRIVILEGES
314
312
401
REUSE CONNECTION COMMAND
404
ACCEPTANCE NOTICE
406
DECLINE REUSE OF CONNECTION FOR USER
408
COPY OF USER COMMAND
410
DECLINE EXECUTION OF USER COMMAND
412
RESULTS OF EXECUTED USER COMMAND
CONNECTION 302
NETWORK 118
MIDDLEWARE SERVER 104
CODE 105
CODE 105C
CODE 105D
CODE 105E
402
USER COMMAND TO BE EXECUTED

FIG. 4

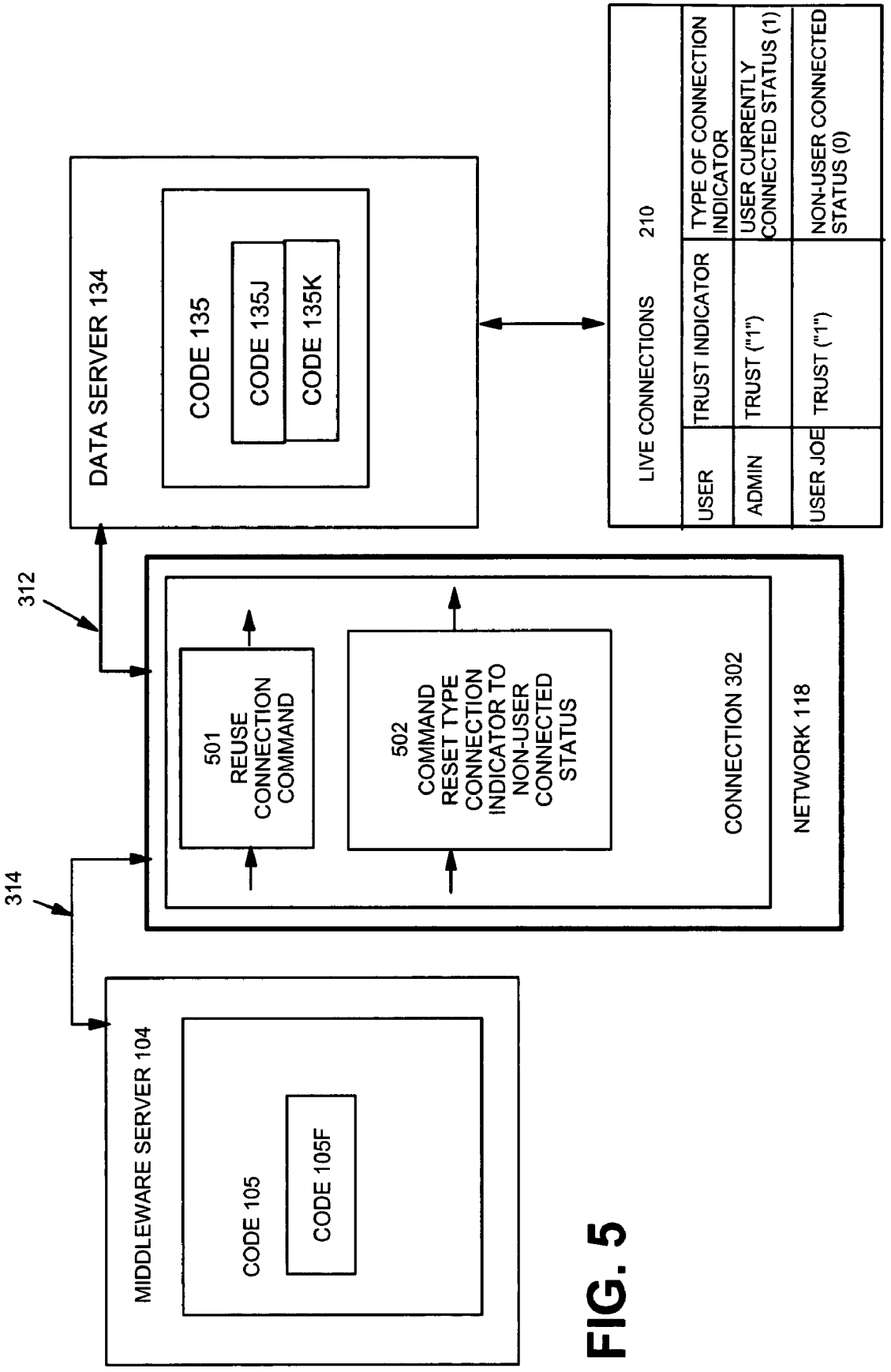

DATA SERVER 134
CODE 135
CODE 135J
CODE 135K
LIVE CONNECTIONS 210
USER
ADMIN
USER JOE
TRUST INDICATOR
TRUST ("1")
TRUST ("1")
TYPE OF CONNECTION INDICATOR
USER CURRENTLY CONNECTED STATUS (1)
NON-USER CONNECTED STATUS (0)
312
314
501 REUSE CONNECTION COMMAND
502 COMMAND RESET TYPE CONNECTION INDICATOR TO NON-USER CONNECTED STATUS
CONNECTION 302
NETWORK 118
MIDDLEWARE SERVER 104
CODE 105
CODE 105F

FIG. 5

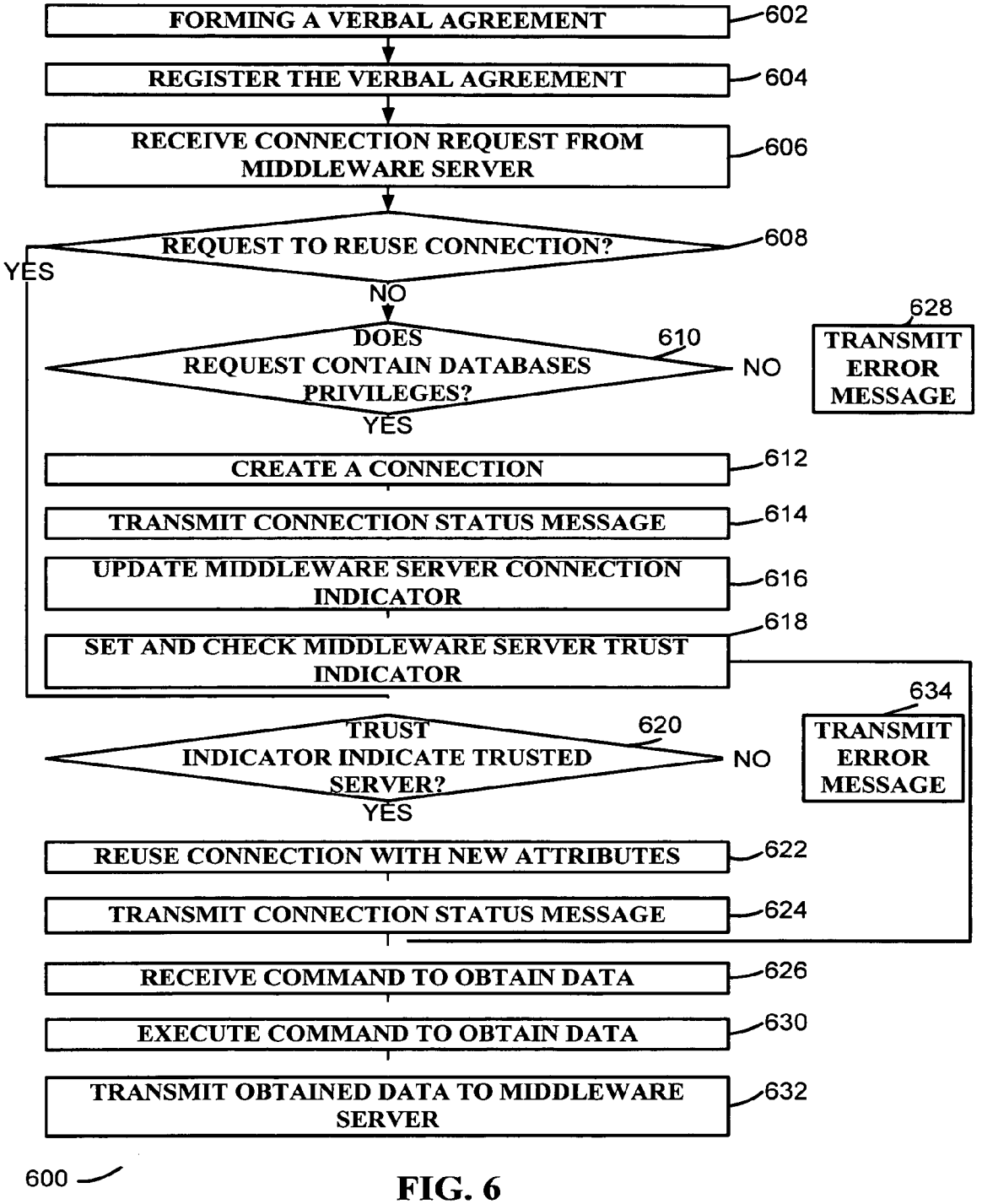

FORMING A VERBAL AGREEMENT
602
REGISTER THE VERBAL AGREEMENT
604
RECEIVE CONNECTION REQUEST FROM MIDDLEWARE SERVER
606
REQUEST TO REUSE CONNECTION?
608
YES
NO
DOES REQUEST CONTAIN DATABASES PRIVILEGES?
610
NO
YES
628
TRANSMIT ERROR MESSAGE
CREATE A CONNECTION
612
TRANSMIT CONNECTION STATUS MESSAGE
614
UPDATE MIDDLEWARE SERVER CONNECTION INDICATOR
616
SET AND CHECK MIDDLEWARE SERVER TRUST INDICATOR
618
TRUST INDICATOR INDICATE TRUSTED SERVER?
620
NO
YES
634
TRANSMIT ERROR MESSAGE
REUSE CONNECTION WITH NEW ATTRIBUTES
622
TRANSMIT CONNECTION STATUS MESSAGE
624
RECEIVE COMMAND TO OBTAIN DATA
626
EXECUTE COMMAND TO OBTAIN DATA
630
TRANSMIT OBTAINED DATA TO MIDDLEWARE SERVER
632
600

FIG. 6

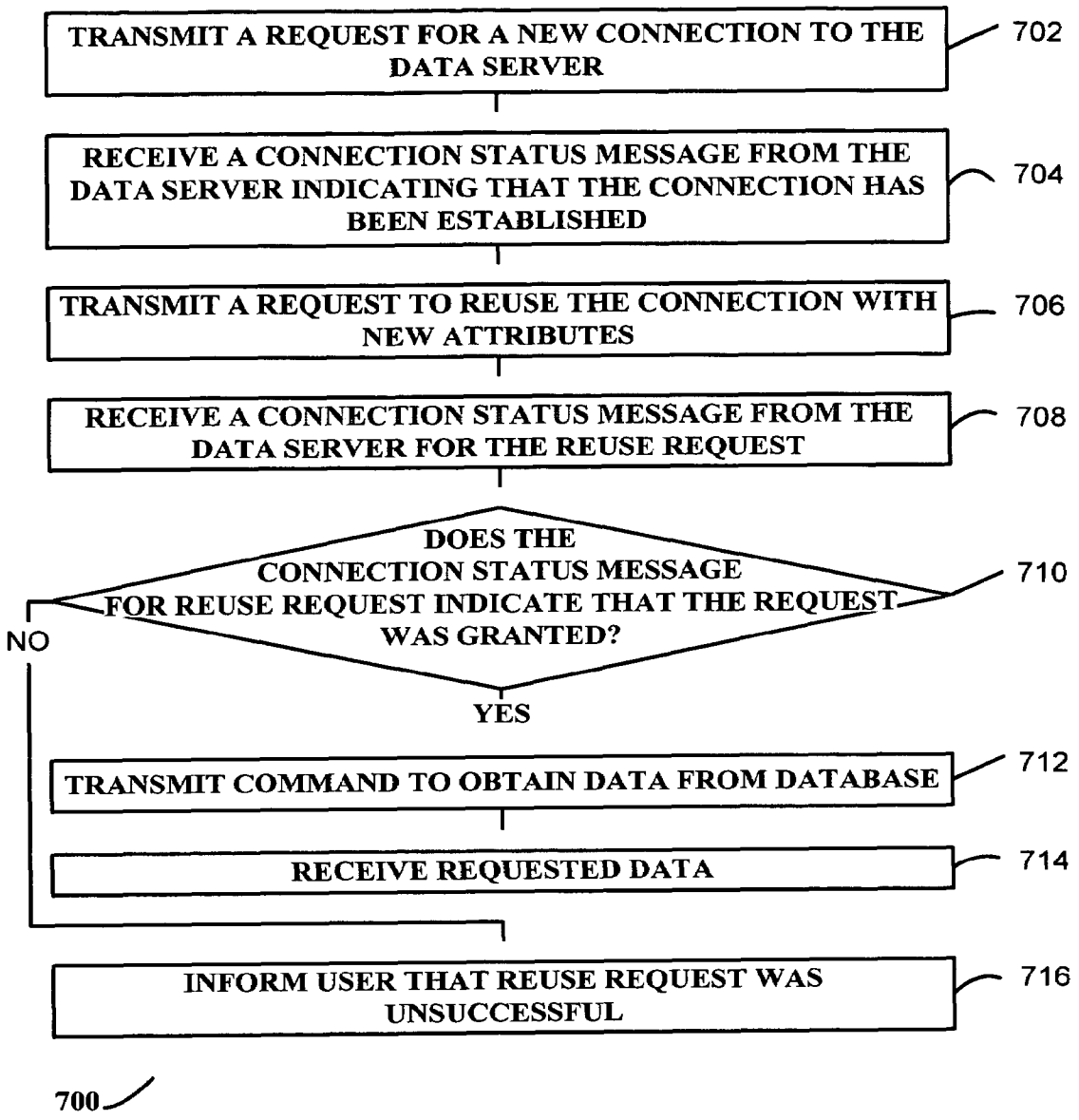

TRANSMIT A REQUEST FOR A NEW CONNECTION TO THE DATA SERVER
702
RECEIVE A CONNECTION STATUS MESSAGE FROM THE DATA SERVER INDICATING THAT THE CONNECTION HAS BEEN ESTABLISHED
704
TRANSMIT A REQUEST TO REUSE THE CONNECTION WITH NEW ATTRIBUTES
706
RECEIVE A CONNECTION STATUS MESSAGE FROM THE DATA SERVER FOR THE REUSE REQUEST
708
DOES THE CONNECTION STATUS MESSAGE FOR REUSE REQUEST INDICATE THAT THE REQUEST WAS GRANTED?
710
NO
YES
TRANSMIT COMMAND TO OBTAIN DATA FROM DATABASE
712
RECEIVE REQUESTED DATA
714
INFORM USER THAT REUSE REQUEST WAS UNSUCCESSFUL
716
700

FIG. 7

METHOD FOR ESTABLISHING A TRUSTED RELATIONSHIP BETWEEN A DATA SERVER AND A MIDDLEWARE SERVER

FIELD OF THE INVENTION

The present invention relates to the field of establishing a trusted relationship between a data server and a middleware server.

BACKGROUND

Access to sensitive data in a database is often managed by relying on the use of user identifications and passwords. If a user desires access to data in the database, a user id and password are often checked to determine if the user is registered to access data from the database. If the user is registered and the correct password has been provided then a connection with the database may be established.

Frequently, access to databases relying on user ids and passwords originate from a few primary locations. However, in such a case multiple user ids may access this data from the same location. Since these locations may be known and trusted, there may not be a requirement to authenticate every different user id and password for these locations.

SUMMARY

In accordance with an aspect of the present invention there is provided for a data server of a data processing system operably coupled to a database, a method of managing a connection with a middleware server, the middleware server sending a request for a connection to the data server, the request comprising request attributes, the method comprising: instituting a set of attributes identifying trusted middleware servers with the data server; establishing a connection with the middleware server based on a request therefrom; and establishing a trust indicator for the connection according to a trust status identified by the set of attributes for the middleware server.

In accordance with an aspect of the present invention there is provided for a middleware server of a data processing system, a method of establishing a connection with a data server operably coupled to a database, the method comprising: transmitting a connection request to the data server, the connection request having request attributes including identifying the connection request as being for a new connection or reuse of an existing connection with different connection request attributes; and receiving a connection status message from the data server indicating a status of the connection request.

In accordance with an aspect of the present invention there is provided for a data server of a data processing system operably coupled to a database, a computer program product for managing a connection with a middleware server, the middleware server sending a request for a connection to the data server, the request comprising request attributes, the computer program product comprising: a computer readable medium for tangibly transporting computer executable code to the middleware server, the computer executable code comprising: code for instituting a set of attributes identifying trusted middleware servers with the data server; code for establishing a connection with the middleware server based on a request therefrom; and code for establishing a trust indicator for the connection according to a trust status identified by the set of attributes for the middleware server.

In accordance with an aspect of the present invention there is provided for a middleware server of a data processing system, a computer program product for establishing a connection with a data server operably coupled to a database, the computer program product comprising: a computer readable medium for tangibly transporting computer executable code to the middleware server, the computer executable code comprising: code for transmitting a connection request to the data server, the connection request having request attributes including identifying the connection request as being for a new connection or reuse of an existing connection with different connection request attributes; and code for receiving a connection status message from the data server indicating a status of the connection request.

A data server of a data processing system is operably coupled to a database and in communication with a middleware server. A connection between the data server and the middleware server is established and managed. A set of attributes identifying trusted middleware servers is instituted with the data server. The middleware server transmits a connection request to the data server. The connection request has request attributes including identifying the connection request as being for a new connection or reuse of an existing connection with different connection request attributes. A connection with the middleware server is established by the data server based on the connection request. A connection status message is received by the middleware server from the data server indicating a status of the connection request. A trust indicator for the connection is established at the data server according to a trust status identified by the set of attributes for the middleware server.

Other aspect and features of the present invention will become apparent to those ordinarily skilled in the art upon review of the following description of embodiments of the invention in conjunction with the accompanying figures.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be described in conjunction with the drawings in which:

FIGS. **1**A and **1**B show a data server operatively coupled to a middleware server;

FIG. **2** shows another middleware server attempting to negotiate data with the data server of FIG. **1**;

FIG. **3** shows the middle ware server of FIG. **1** attempting to negotiate data with the data server of FIG. **1**;

FIG. **4** shows a response taken by the middleware server and the data server of FIG. **1** when a user command requests a copy of data from the data server **134**;

FIG. **5** shows interaction between the middleware server and the data server of FIG. **1** when another user attempts to send a command requesting data servable by the data server;

FIG. **6** illustrate a method of managing connections with the middleware server for the data server; and

FIG. **7** illustrates a method of establishing a connection with the data server for the middleware server.

DETAILED DESCRIPTION

FIG. **1**A shows two data processing systems (A **100** and B **130**) in operable communication via network **118**. Both data processing systems **100** and **130** contains a bus **108** that operatively couples a central processing unit (CPU) **109**, an input/output interface **110** and a memory **102**/**132**. The input/output interface **110** manages communications between the

bus **108** and a display **112**, a keyboard **114**, a disc **116** and the network **118** for each of the data processing systems **100** and **130**.

The memory **132** of data processing system B **130** includes a data server **134**, a database **136** operatively coupled to the data server **134**, and an operating system **133**. The data server **134** may be, for example, an information retrieval system of a database management system. The data server **134** includes computer executable code **135** with a collection of modules **135**A to **135**K. These modules **135**A to **135**K perform functions (when compiled and executed) that compose a data server method. The functions of the data server method may include using the network **118** to communicate with data processing system A **100**. In an alternate embodiment, the database **136** may be located in another data processing system, in which case, the data processing system B **130** may use the network **118** for communication with the database **136**.

The memory **102** of the data processing system A **100** includes a middleware server **104** and an operating system **103**. The middleware server **104** includes computer executable code **105** with a collection of modules **105**A to **105**K. These modules **105**A to **105**K perform functions (when compiled and executed) that compose a middleware server method. The functions of the modules **105**A to **105**K may include directing the middleware server **104** to negotiate for data with the data server **134**.

The data server **134** of the data processing system B **130** acts as an intermediary between the database **136** and the middleware server **104**.

A data server administrator **152** administers the data server **134** according to a verbal agreement **154**. A middleware server administrator **150** administers the middleware server **104** according to the verbal agreement. The verbal agreement **154** contains an agreement between the data server administer **152** and the middleware server administer **150** regarding the characteristics of connections and data transfers between the middleware server **104** and the data server **134**. For example, the verbal agreement **154** may set forth that the data server **134** is to be set up such that connection requests from the middleware server **104** received thereat are treated by the database **136** as trusted connection. To perform such an exemplary set up, the data server administrator **152** registers (in the database **136**, for example) connection trust attributes that are associated with the trusted connections. The connection trust attributes may include a user id associated with a trusted connection, an IP (internet protocol) address of data processing system from which connection requests are received, or other attributes. The middleware server **104** includes user identification attributes **105** for the middleware server administrator **140**.

It will be appreciated that the middleware server **104**, the data server **134** and the database **136** may be present on the same or on different data processing systems.

FIG. **1**B shows an alternative to storing the database **136** in the memory **132** of the data processing system B **130** of FIG. **1**A. The database **136** maybe stored in a memory **152** of a data processing system C **150**. The data processing systems A **100**, B **130** and C **152** are in operable communication via the network **188**.

FIG. **2** illustrates another middleware server **204** in communication with the data server **134** via the network **118**. The middleware server **204** contains code **208** with a module **208**A. The middleware server **204** is located in a memory **202** of a data processing system **201**.

The data processing system **201** includes known modules that facilitate communication via the network **118** and is indicated as a connection line **216**. Such a connection line **212** is also used to connect the data processing system B **130** with the network **118**. The connection lines **216** and **212** are used for establishing a connection **206** (via the network **118**) between the data processing system B **130** and the data processing system **201**.

The middleware server **204** has not been set up as being trusted on the data server **134** by the data server administrator **152**. This may be because an agreement was not previously set between administrators of the data server **134** and the middleware server **204** to govern interactions between these two servers **134** and **204**.

The middleware server **204** is attempting to negotiate to obtain data through the data server **134**. The code **208** directs a CPU (not illustrated) of the data processing system **201** to establish the connection **206** with the data server **134**. Once the connection **206** is established, the code **208** directs the data processing system **201** to issue a request **214** for requesting access to the data associated with the data server **134**. The request **214** is sent to the data server **134** via the connection line **216** through the connection **206** and over to the connection line **212**.

A list **210** is stored in the memory **132** of the data processing system A **130** indicating currently active connections established with the data server **134**. The code **135**A directs the data processing system B **130** to update the list **210** in response to the data processing system B **130** establishing a connection with the middleware server **204**. Once the connection **206** is set up, the code **135**B directs the data processing system B **130** to set a trust indicator in the list **210** to "do not trust" (for example, a bit may be used and set to a value of "0" for this case) because the middleware server **204** has not bee previously registered with the data server **134** as a trusted entity.

The decision to trust or not to trust a requesting middleware server is performed by the data server **134** on the basis of verbal agreements **154** between the database **136** and various middle ware servers that have been registered with the data server **134**. Such verbal agreements **154** are registered with the data server **134** by the data server administrator **152** to provide an indication of connections that are to be trusted. This information may be stored in a table that the data server **134** can search each time a request connection is received.

For each connection request received, the data server **134** compares attributes of the connection request (e.g. user id, IP address, etc.) with information stored in the database **136** about the connections that are to be trusted. If there is a match then the current connection is marked as a trusted connection; otherwise, the connection is marked as untrusted.

The data server **134** will establish a connection with the middleware server **204** based on the request; however, that connection will be marked as not trusted because the middleware server **204** has not been registered as trusted on the data server **134**. The data server **134** will continue to honor requests from the middleware server **204** but since the connection between these two is not trusted the data server **134** will reject a request from the middleware server **204** to reuse the connection under a different user id without supplying a password. On the other hand, a middleware server that is registered as being trusted with the data server **134** will have requests to reuse the existing connection under a different user granted without requiring that a password be supplied.

FIG. **3** illustrates a connection between the middleware server **104** and the data server **134** of FIG. **1**. The middleware server **104** has a connection line **312** with the network **118** and the data server **134** has a connection line **314** with the network **118**. The memory **132** of the data processing system B **130** may include a list **310** of trusted middle ware servers in

addition to the data server **134** and the list of current active connections **210**. The list **310** of trusted middleware server may also be derived when examining a request by looking at the list **210** and selecting those connections that have a positive trust indicator.

The list **210** of current active connections includes an indication as to whether or not the connection is trusted. A connection is trusted when the data server **134** determines that the connection's source attributes match the attributes of a connection source (i.e. middle ware server) registered in the database **136** as to be trusted.

The middleware server **104** attempts to make a connection with the data server **134**. The code **105** A directs the middleware server **104** to establish a connection **302** with the data server **134**.

The data processing system **100** includes known modules that facilitate communication via the network **118** and is indicated as a connection line **312**. Such a connection line **314** is also used to connect the data processing system B **130** with the network **118**. The connection lines **312** and **314**re used for establishing a connection **302** (via the network **118**) between the data processing system B **130** and the data processing system **100**.

A request **306** to establish a connection along with a copy **308** of the user id **106** currently associated with the middleware server **104** are sent as a package **304** from the middleware server **104** to the data server **134** via the network **118**. There are two possible scenarios for processing of this request by the data server **134**: either this is a new connection between the middleware server **104** and the data server **134** or a connection already exists between these two parties and the request contains a request to maintain the connection therebetween using a different user id (and possibly a password).

If the package **304** is for a request for a new connection, then the data server **134** receives the request and authenticates the user id and the password before the connection is established. As part of the authentication process, the data server updates the list **210** of current active connections. The data server **134** then examines attributes of the connection with the middleware server **104** and if such attributes match attributes in the database **136** of a trusted server then the middleware server **104** is identified as being trusted and marks the connection as trusted. Once this is complete, the middleware server **104** can start requesting services from the database **136** through the data server **134** via this connection.

If the package is for a request to maintain a connection with a different user id, then the data server **134** receives the request and examines the list **210** to determine if the middleware server **104** is a trusted connection. If the existing connection between the middleware server **104** and the data server **134** is trusted then the current connection between the parties is maintained with the different user id without requiring a password associated with the different user id; otherwise, a password is required and the connection can be broken.

The connection source attributes in the database **136** may indicate that all connections from a particular source are to be trusted, irregardless of the user id. Alternatively, the connection source attributes may indicate only specific user ids that may be interchanged on a trusted connection without the requirement of a password, other user ids from the same source may require passwords.

FIG. **4** illustrates a situation when a user command **402** is received by the middleware server **104** requesting a copy of data from the data server **134**.

In response to receiving the user command **402**, the code **105**C directs the middleware server **104** to transmit a reuse connection command **401** to the data server **134** via the established connection **302**. The connection **302** shows a copy of the reuse connection command **401** which is then received by the data server **104**.

In response to the data server **134** receiving the reuse connection command **401**, the code **135**H directs the data server **134** to determine whether to reuse the established connection **302** for executing the user command **402** received by the middleware server **104**. The code **135**H may direct the data server **134** to issue a notice indicating that the established connection **302** may be reused for executing the user command **402** submitted by the middleware server **104** if the trusted indicator (as shown in table **210**) indicates that the middleware server **104** may be trusted. If the middleware server **104** may be trusted, the code **135**H may direct the data server **134** to transmit an acceptance notice **404** to the middleware server **104** via connection **302**. The code **135**H may direct the data server **134** to decline executing the user command **402** received by the middleware server **104** if the trusted indicator (as shown in table **210**) indicates that the middleware server **134** may not be trusted. If the middleware server **104** is not to be trusted, the code **135**H may direct the data server **134** to transmit a decline notice (to the middleware server **104**) for declining the reuse of the connection **302** for the user command **402**.

In response to receiving the acceptance notice **404**, the code **105**D may direct the middleware server **104** to transmit the user command **402** to the data server **134** (via connection **302**); thus, the connection **302** is reused for transmitting the user command to the data server **134**. In response to receiving the decline notice, the code **105**D may direct an error message (not shown) to the user who submitted the user command **402** indicating that the user command **408** for requesting data access was declined by the data server **134**.

In response to the data server **134** receiving a copy of the user command **408** from the middleware server **104**, the code **1351** directs the data server **134** to receive the copy of the user command **408**, and then to execute the user command **408**.

In an alternative, before the data server **134** executes the user command **408**, the code **135**I may direct the data server **143** to determine whether the user associated with the user command **408** has predetermined data access privileges (for accessing the data being requested) that were previously established with the data server **134**. For example, the data server **134** may decline execution of the user command **408** because the data server **134** determines that the user as no predetermined access rights established for accessing that data identified in the user command **408**. In this case, the data server **134** transmits a decline execution notice **410** to the middleware server **104**. For the case when the data server **134** determines that the user is associated with access privileges with the data, the data server **134** may execute the user command **408** to access the data stored in the database **136**, then the data server **134** transmits the accessed data **412** via connection **302** over to the middleware server **104**.

In response to receiving the decline notice **410** declining access to data, the code **105**E directs the middleware server **104** to transmit an error message (not shown) to the user. In response to receiving the accessed data **412**, the code **105**E directs the middleware server **104** to transmit the accessed data **412** to the user.

FIG. **5** shows an interaction between the data server **134** and the middleware server of FIG. **1** when another user attempts to send a command from the middleware server **104** for accessing data via by the data server **134**. The code **105** includes the code **105**F. The code **135** includes the code **135**J and the code **135**K.

In response to receiving a release signal from the user, the code **105**F directs the middleware server **104** to transmit a type of connection reset command **501** to the data server **134**. The reused connection command **501** is shown in the connection **302**.

In response to receiving the reused connection command **501** via the established connection **302**, the code **135**J directs the data server **134** to set a type of connection indicator of user to indicate that the user is currently connected. For example, the data server **134** may set the user ID to the name of the current user of the connection. Once the user has completed using the connection **302**, the user may wish to either request more data from the data server **134** or reset the type of connection indicator which permits other users to interact with the data server **134**.

In response to receiving a release indicator from the user, the code **105**F directs the middleware server **104** to transmit a type of connection reset command **502** to the data server **134** via the established connection **302**.

The code **135**J, in response to the data server **134** receiving the type of connection reset command **502** via the connection **502**, directs the data server **134** to permit another user to use the connection **302**. The table **210** contains indications for each user id of the trust and use status; for example, the non-user status connection indicator is "1" (which indicates the user JOE is reusing the connection **302**), and the user status connection indicator is set to "0" to indicate that the administrator **150** of the MDW_(W) **104** is not using the connection **302**. This arrangement provides a mechanism which permits user JOE exclusive channel to submit user commands to the data server **134**.

In response to receiving the reset command **502** from the middleware server **104** via the connection **302**, the code **135**K directs the data server **134** to set the type of connection indicator to indicate a non-user connection status, which includes setting the currently connected status to of user ID=Admin (the administrator **152**) to "1" (the "1" indicates the administrator has control of the connection **302**), and setting the currently connected status of user JOE to "0" (the "0" indicates that user JOE is no longer the active user using the connection **302**). Now another user of the middleware server may reuse the connection **302**.

FIG. **6** illustrates a method **600** for the data server **134** of managing a connection with a middleware server. A verbal agreement is formed between the middleware server and the data server in step **602**. The verbal agreement indicates whether or not the data server will trust the middleware server and to what degree the middleware server will be trusted. This verbal agreement is then registered with the data server in step **604**.

A connection request is received from the middleware server in step **606**. The connection request includes request attributes such as whether the request is for a new connection or a reuse of an existing connection, a user identification (and possibly password) for a user of the middleware server, an IP address for the middleware server, etc. In step **608** the request attributes are examined to determine if the request is a reuse of an existing connection using different attributes (e.g. different user identification). If the request is not to reuse the connection then it is a request for a new connection.

The request for a new connection is examined in step **610** to determine if the request attributes contain access privileges for the database (e.g. does the user identification and password match a user id and password registered in the database). If the request contains access privileges, then a connection between the data server and the middleware server is created in step **612**. A connection status message is transmitted to the middleware server in step **614** indicating that the connection was established. A connection indicator is updated in step **616** to indicate that the middleware server is connected with the data server. A trust indicator is then set and checked for the middleware server in step **618**. Based on attributes of the middleware server in the request (e.g. IP address) and attributes of servers that can be trusted (as found in the registered verbal agreement), the trust indicator is set as 'trust' or 'not trust' for the middleware server.

If the connection request for a new connection does not contain database access privileges then an error message is transmitted to the middleware server in step **628**.

If the connection request is to reuse a connection then the trust indicator for the middleware server is examined in step **620**. If the trust indicator indicates that the middleware server is not a trusted server then an error message is transmitted to the middleware server in step **634**.

If the trust indicator indicates that the middleware server is a trusted server then the connection may be reused with new attributes. These new attributes are set for the connection in step **622**. A connection status message is transmitted to the middleware server in step **624** indicating that the connection is being reused with the new attributes.

After a new connection has been established or the existing connection is set up to be reused, a command to obtain data from the data base is received from the middleware server in step **626**. The command to obtain data is executed in step **630** and the obtained data is transmitted to the middleware server in step **632**.

FIG. **7** illustrates a method **700** of establishing a connection with the data server by the middleware server. A request for a new connection is transmitted from the middleware server to the data server instep **702**. A connection status message is received from the data server in step **704** indicating whether or not the connection has been established.

After a connection has been established a request to reuse the connection with different attributes **9**e.g. different user id) is transmitted to the data server in step **706**. A connection status message is received form the data server in step **708** indicating whether or not the request to reuse the connection was granted.

The connection status message is examined in step **710** to determine if the request to reuse the connection was granted. If the request was not granted then a user is informed in step **716** that the request was unsuccessful.

If the request was successful then a command to obtain data from the database is transmitted to the data server in step **712**. The requested data is received form the data server in step **714**.

The detailed description of the embodiments of the present invention does not limit the implementation of the embodiments to any particular computer programming language. The computer program product may be implemented in any computer programming language provided that the OS (Operating System) provides the facilities that may support the requirements of the computer program product. A preferred embodiment is implemented in the C or C++ computer programming language (or may be implemented in other computer programming languages in conjunction with C/C++). Any limitations presented would be a result of a particular type of operating system, computer programming language, or data processing system and would not be a limitation of the embodiments described herein.

It will be appreciated that the elements described above may be adapted for specific conditions or functions. The concepts of the present invention can be further extended to a variety of other applications that are clearly within the scope

of this invention. Having thus described the present invention with respect to preferred embodiments as implemented, it will be apparent to those skilled in the art that many modifications and enhancements are possible to the present invention without departing from the basic concepts as described in the preferred embodiment of the present invention.

The invention claimed is:

1. For a data server of a data processing system operably coupled to a database, a method of managing a connection with a middleware server, the middleware server sending a request for a connection to the data server, the request comprising request attributes, the method comprising:

storing a set of attributes identifying middleware servers trusted by the data server;

establishing a connection between the middleware server and the data server based on a request, having connection request attributes, received from the middleware server; and

setting a trust indicator for the connection, according to a trust status determined by comparing the set of attributes identifying the middleware server to the received connection request attributes, the trust status indicating whether the connection is one of a trusted connection and a non-trusted connection,

wherein if the connection between the middleware server and the data server is a trusted connection, the data server permits use of the connection by the middleware server when a first user is connected to the middleware server and permits reuse of the connection by the middleware server when a second user, different from the first user, is connected to the middleware server without requiring authentication of the second user.

2. The method of claim **1**, wherein the step of establishing a connection comprises:

updating a connection indicator after the connection has been established to indicate that the connection has been established.

3. The method of claim **2**, wherein the step of setting a trust indicator comprises:

determining if the trust indicator is set for the connection; and

if the trust indicator is set, determining whether the connection is trusted after the connection indicator indicates the connection is established.

4. The method of claim **1** wherein the step of storing comprises:

forming an agreement between the data server and the middleware server containing the set of attributes; and

registering the set of attributes with the data server.

5. The method of claim **1** wherein the step of establishing a connection comprises:

receiving a request from the middleware server to establish a connection therebetween; and

determining whether the request attributes indicate a request for a new connection or a reuse of an existing connection with different request attributes.

6. The method of claim **5** wherein the step of establishing a connection further comprises:

determining if the request attributes include access privileges for the database if the request attributes indicate a request for a new connection;

creating the connection if the request attributes include access privileges; and

transmitting a connection status message to the middleware server indicating that the connection has been established.

7. The method of claim **5** wherein the request attributes comprises a user identification and the different request attributes comprises a different user identification.

8. The method of claim **7**, wherein the step of establishing the connection further comprises:

determining whether the existing connection can be reused based on the trust indicator of the existing connection.

9. The method of claim **8**, wherein the step of determining whether the existing connection can be reused comprises:

transmitting a connection status message to the middleware server indicting that the connection may be reused if the trust indicator indicates that the middleware server is trusted.

10. The method of claim **8**, wherein the step of determining whether the existing connection can be reused comprises:

transmitting a connection status message to the middleware server indicating that the existing connection may not be reused if the trust indicator indicates that the middleware server is not trusted.

11. The method of claim **1** further comprising:

receiving a command via the connection for obtaining data in the database from the middleware server; and

executing the command in the request if the trust indicator for the middleware server indicates that the middleware server is trusted.

12. The method of claim **11**, wherein the step of executing comprises:

transmitting a decline execution notice to the middleware server if the request attributes do not include access privileges for the data identified in the command; and

transmitting obtained data to the middleware server in response to the command if the request attributes include access privileges for the data identified in the command.

13. The method of claim **1**, wherein the middleware server receives the request for connection to the data server from a user connected to the middleware server over a network.

14. The method of claim **1**, wherein the connection request attributes comprise attributes of a user connected to the middleware server and attributes of the middleware server.

15. The method of claim **1**, wherein the trust indicator is stored at the data server.

16. The method of claim **1**, wherein the storing the set of attributes identifying middleware servers trusted by the data server occurs prior to the establishing the connection between the middleware server and the data server based on the request.

17. For a middleware server of a data processing system, a method of establishing a connection with a data server operably coupled to a database, the method comprising:

transmitting a connection request to the data server, the connection request having request attributes that identify the connection request as being one of a new connection and reuse of an existing connection having different connection request attributes; and

receiving a connection status message from the data server indicating a status, when the data server determines that the middleware server is one of a trusted middleware server and a non-trusted middleware server by comparing the request attributes to a stored set of attributes identifying the middleware server, of the connection as being one of a trusted connection and a non-trusted connection,

wherein if the connection between the middleware server and the data server is a trusted connection, the data server permits use of the connection by the middleware server when a first user is connected to the middleware

server and permits reuse of the connection by the middleware server when a second user, different from the first user, is connected to the middleware server without requiring authentication of the second user.

18. The method of claim **17**, wherein the step of transmitting a connection request comprises:

transmitting the connection request with the request to reuse the existing connection to the data server via the existing connection,

wherein the request attributes comprises a user identification and the different request attributes comprises a different user identification than the existing connection.

19. The method of claim **17**, further comprising:

transmitting a command for obtaining data in the database to the data server if the connection status message indicates that the connection has been established.

20. The method of claim **19** further comprising:

receiving obtained data from the data server in response to the command.

21. For a data server of a data processing system operably coupled to a database, a computer program product having computer executable codes embodied on a computer-readable storage medium for managing a connection with a middleware server, the middleware server sending a request for a connection to the data server, the request comprising request attributes, the computer program product comprising:

code storing a set of attributes identifying middleware servers trusted by the data server;

code establishing a connection between the middleware server and the data server based on the request received from the middleware server; and

code setting a trust indicator for the connection according to a trust status determined by comparing the set of attributes identifying the middleware server to the received connection request attributes, the trust status indicating whether the connection is one of a trusted connection and a non-trusted connection,

wherein if the connection between the middleware server and the data server is a trusted connection, the data server permits use of the connection by the middleware server when a first user is connected to the middleware server and permits reuse of the connection by the middleware server when a second user, different from the first user, is connected to the middleware server without requiring authentication of the second user.

22. The computer program product of claim **21**, wherein the code establishing a connection comprises:

code updating a connection indicator after the connection has been established to indicate that the connection has been established.

23. The computer program product of claim **22**, wherein the code setting a trust indicator comprises:

code determining if the trust indicator is set for the connection; and

code, if the trust indicator is set, determining whether the connection is trusted after the connection indicator indicates the connection is established.

24. The computer program product of claim **21**, wherein the code storing comprises:

code forming an agreement between the data server and the middleware server containing the set of attributes; and

code registering the set of attributes with the data server.

25. The computer program product of claim **21**, wherein the code establishing a connection comprises:

code receiving a request from the middleware server to establish a connection therebetween; and

code determining whether the request attributes indicate a request for a new connection or a reuse of an existing connection with different request attributes.

26. The computer program product of claim **25**, wherein the code establishing a connection further comprises:

code determining if the request attributes include access privileges for the database if the request attributes indicate a request for a new connection;

code creating the connection if the request attributes include access privileges; and

code transmitting a connection status message to the middleware server indicating that the connection has been established.

27. The computer program product of claim **25**, wherein the request attributes comprises a user identification and the different request attributes comprises a different user identification.

28. The computer program product of claim **27**, wherein the code establishing the connection further comprises:

code determining whether the existing connection can be reused based on the trust indicator of the existing connection.

29. The computer program product of claim **28**, wherein the code determining whether the existing connection can be reused comprises:

code transmitting a connection status message to the middleware server indicting that the connection may be reused if the trust indicator indicates that the middleware server is trusted.

30. The computer program product of claim **28**, wherein the code determining whether the existing connection can be reused comprises:

code transmitting a connection status message to the middleware server indicating that the existing connection may not be reused if the trust indicator indicates that the middleware server is not trusted.

31. The computer program product of claim **21**, further comprising:

code receiving a command via the connection for obtaining data in the database from the middleware server; and

executing the command in the request if the trust indicator for the middleware server indicates that the middleware server is trusted.

32. The computer program product of claim **31**, wherein the step of executing comprises:

code transmitting a decline execution notice to the middleware server if the request attributes do not include access privileges for the data identified in the command; and

code transmitting obtained data to the middleware server in response to the command if the request attributes include access privileges for the data identified in the command.

33. For a middleware server of a data processing system, a computer program product having computer executable codes embodied on a computer-readable storage medium for establishing a connection with a data server operably coupled to a database, the computer program product comprising:

code transmitting a connection request to the data server, the connection request having request attributes that identify the connection request as being for one of a new connection and reuse of an existing connection having different connection request attributes; and

code receiving a connection status message from the data server indicating a status, when the data server determines that the middleware server is one of a trusted middleware server and a non-trusted middleware server by comparing the request attributes to a stored set of

attributes identifying the middleware server, of the connection as being one of a trusted connection and a non-trusted connection,

wherein if the connection between the middleware server and the data server is a trusted connection, the data server permits use of the connection by the middleware server when a first user is connected to the middleware server and permits reuse of the connection by the middleware server when a second user, different from the first user, is connected to the middleware server without requiring authentication of the second user.

34. The computer program product of claim **33**, wherein the code transmitting a connection request comprises:

code transmitting the connection request with the request to reuse the existing connection to the data server via the existing connection,

wherein the request attributes comprises a user identification and the different request attributes comprises a different user identification than the existing connection.

35. The computer program product of claim **33**, further comprising:

code transmitting a command for obtaining data in the database to the data server if the connection status message indicates that the connection has been established.

36. The computer program product of claim **35**, further comprising:

code receiving obtained data from the data server in response to the command.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 7,647,626 B2
APPLICATION NO. : 11/008507
DATED : January 12, 2010
INVENTOR(S) : Bird et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

On the Title Page:

The first or sole Notice should read --

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1047 days.

Signed and Sealed this

Sixteenth Day of November, 2010

David J. Kappos

David J. Kappos
Director of the United States Patent and Trademark Office

US007865521B2

(12) **United States Patent**
Bird et al.

(10) **Patent No.: US 7,865,521 B2**
(45) **Date of Patent: *Jan. 4, 2011**

(54) **ACCESS CONTROL FOR ELEMENTS IN A DATABASE OBJECT**

(75) Inventors: **Paul Miller Bird**, Markham (CA); **Walid Rjaibi**, Markham (CA)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 263 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **11/299,857**

(22) Filed: **Dec. 12, 2005**

(65) **Prior Publication Data**

US 2007/0136291 A1 Jun. 14, 2007

(51) **Int. Cl.**
G06F 7/00 (2006.01)
G06F 17/30 (2006.01)

(52) **U.S. Cl.** **707/781**; 707/783; 707/999.1; 707/999.9

(58) **Field of Classification Search** 707/1–10, 707/781, 783, 999.1, 999.9
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,815,574	A	9/1998	Fortinsky	
6,026,388	A	2/2000	Liddy et al.	
6,085,191	A *	7/2000	Fisher et al.	707/9
6,308,273	B1	10/2001	Goertzel et al.	
6,321,235	B1	11/2001	Bird	707/203
6,321,334	B1	11/2001	Jerger et al.	
6,424,974	B1	7/2002	Cotner et al.	707/103
6,487,552	B1 *	11/2002	Lei et al.	707/4
6,643,633	B2	11/2003	Chau et al.	707/1
6,721,727	B2	4/2004	Chau et al.	707/3
7,133,875	B1 *	11/2006	Chatterjee et al.	707/999.102
2003/0046550	A1 *	3/2003	Carroll et al.	713/185
2004/0139043	A1 *	7/2004	Lei et al.	707/1
2005/0246338	A1	11/2005	Bird	
2007/0033196	A1 *	2/2007	Moore	707/10
2007/0038596	A1 *	2/2007	Pizzo et al.	707/2
2008/0275880	A1	11/2008	Bird et al.	

OTHER PUBLICATIONS

Panagiotis Katsaros, "On the Design of Access Control to Prevent Sensitive Information Leakage in Distributed Object Systems: A Colored Petri Net Based Model", SpringerLink Contemporary, Oct. 11, 2005, vol. 3761. Download: http://www.springerlink.com/content/5p71w09j9rlepaye/fulltext.pdf.*

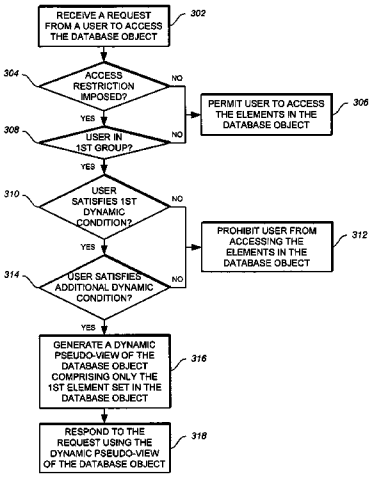
(Continued)

Primary Examiner—John E Breene
Assistant Examiner—Hares Jami
(74) *Attorney, Agent, or Firm*—Terry Carroll; SVL IP Law

(57) **ABSTRACT**

A method, for controlling access to elements in a database object are provided. The method provide for receiving a request from a user to access the database object, determining whether an access restriction is imposed on the database object, and controlling access to the elements in the database object by the user based on the access restriction. The access restriction specifies one or more users to which the access restriction is applicable, defines a dynamic condition the one or more users must satisfy in order to access the database object, and identifies one or more of the elements in the database object accessible to the one or more users when the dynamic condition is satisfied.

30 Claims, 7 Drawing Sheets



OTHER PUBLICATIONS

Hadjiefthymiades, SP et al., "A Generic Framework for the Deployment of Structured Databases on the World Wide Web," Computer Networks and ISDN Systems, May 1996, vol. 28, No. 7-11, 9 pgs.
Zhenchuan, Xu et al., "Dynamic Tuning of XML Storage Schema in VXMLR," IEEE 2003, Proceedings of the Seventh International Database Engineering and Applications Symposium, pp. 1-11.
International Search Report (ISR) dated Jan. 3, 2007, for corresponding foreign Application.
Rakesh Agrawal, et al., "Extending Relational Database Systems To Automatically Enforce Privacy Policies", IBM Almaden Research Center, 650 Harry Road, San Jose, CA, Proceedings of the 21st International Conference on Data Engineering , IEEE, 2005.
Office Action dated Oct. 16, 2006; cited in U.S. Appl. No. 10/837,387.
Final Office Action dated May 4, 2007; cited in U.S. Appl. No. 10/837,387.
Office Action dated Oct. 18, 2007; cited in U.S. Appl. No. 10/837,387.
Office Action dated Apr. 23, 2008; cited in U.S. Appl. No. 10/837,387.
Office Action dated Apr. 10, 2009; cited in U.S. Appl. No. 10/837,387.
Tzelepi, S. et al., "Security of Medical Multimedia," Medical Informatics and the Internet in Medicine, vol. 27, No. 3, Sep. 2002, pp. 169-184.
Damiani, E. et al., "Regulating Access to Semistructured Information on the Web," Information Security for Global Information Infrastructures, Sixteenth Annual Working Conf. on Information Security, Aug. 22-24, 2000, Beijing, China, pp. 351-360.
Duesterwald, E. "A Practical Data Flow Framework for Array Reference Analysis and its use in Optimizations," ACM Sigplan Notices vol. 28, No. 6, Jun. 1993, Proc of the ACM SIGPLAN '93 Conf. on Programming Language Design and Implementation Albuquerque, NM, Jun. 23-25, pp. 68-67.
Low, M. et al., "Fine Grained Object Protection in Unix," Operating Systems Review vol. 27, No. 1, Jan. 1993, pp. 33-50.
Salemi, C. et al., "A Privilege Mechanism for UNIX System V Release 4 Operating Systems," Conf. Proceedings, USENIX, Summer 1992 Technical Conf., San Antonio, Texas, Jun. 8-12, 1992, pp. 235-241.
Leiss, E. et al., "Protecting Statistical Databases by Combining Memoryless Table Restrictions With Randomizations," AFIPS Conf., Proc., vol. 56, 1987 National Computer Conference, Jun. 15-18, 1987, Chicago, Illinois, pp. 591-600.
Damiani, E. et al., "A Fine-Grained Access Control System for XML Documents," ACM Transactions on Information and System Security, vol. 5, No. 2, May 2002, pp. 169-202.
Grimm, R. et al., "Separating Access Control Policy, Enforcement, and Functionality in Extensible Systems," ACM Transactions on Computer Systems, vol. 19, No. 1, Feb. 2001, pp. 36-70.
Coulouris, G. et al., "Security Requirements for Cooperative Work: A Model and Its System Implications," Position Paper for 6th SIGOPS European Workshop, Dagstuhl, Sep. 1994, pp. 184-186.
Wang, Weigang, "Team-and-Role Based Organizational Context and Access Control for Cooperative Hypermedia Environments," Hypertext 99, Darmstadt Germany, Copyright ACM 1999, pp. 37-46.
Böhlen, Michael H. et al., "Temporal Statements Modifiers," ACM Transaction on Database Systems, vol. 25, No. 4, Dec. 2000, pp. 407-456.
Lakshmanan, Laks, V.S. et al., "SchemaSQL-An Extension to SQL for Multidatabase Interoperability," ACM Transactions on Database Systems, vol. 26, No. 4, Dec. 2001, pp. 476-519.
Ng, Wilfred, "An Extension of the Relational Data Model to Incorporate Ordered Domains," ACM Transactions on Database Systems, vol. 26, No. 3, Sep. 2001, pp. 344-383.
Hadjiefthymiades, SP et al., "A Generic Framework for the Deployment of Structured Databases on the World Wide Web," Computer Networks and ISDN Systems, May 1996, vol. 28, No. 7-11, pp. 1139-1148. (Abstract).
Zhenchuan, Xu et al., "Dynamic Tuning of XML Storage Schema in VXMLR," 2003, Proceedings International Database Engineering and Applications Symposium, pp. 76-86. (Abstract).

* cited by examiner

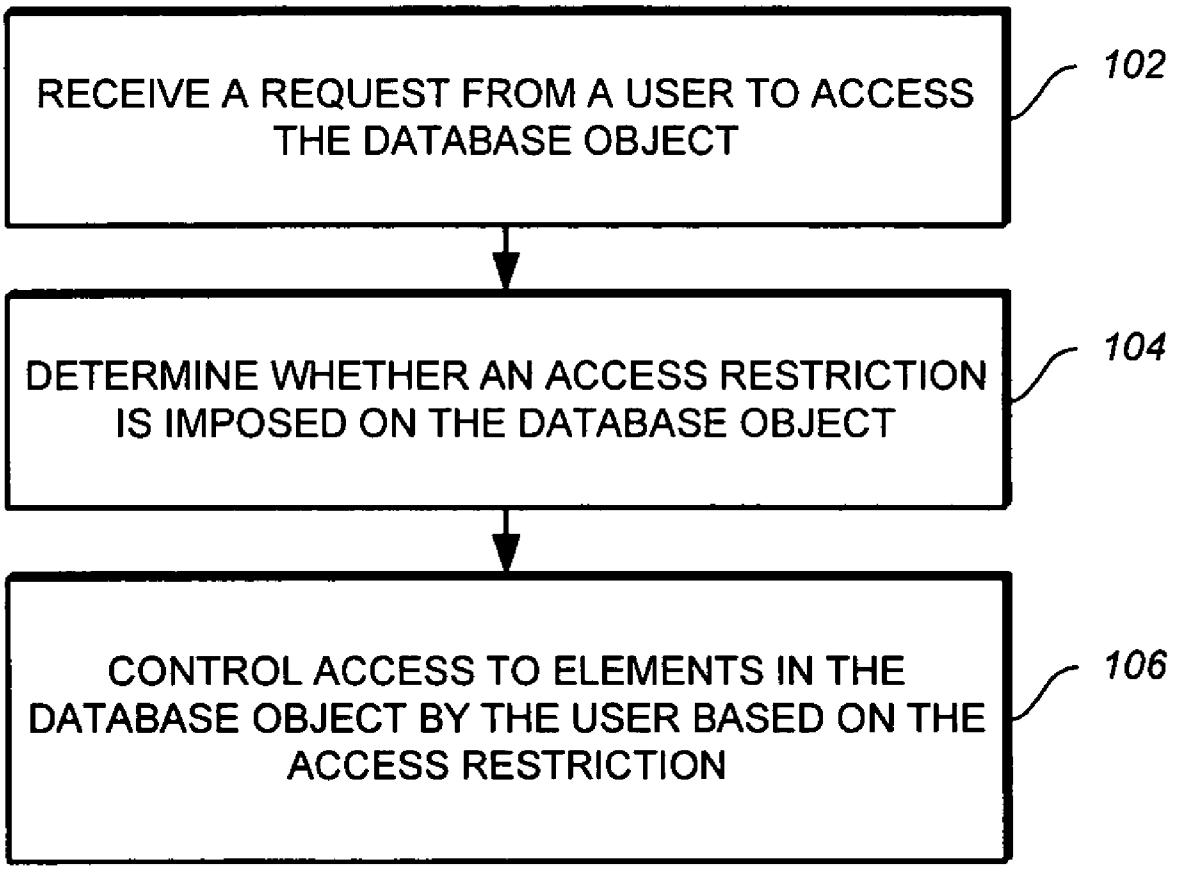

RECEIVE A REQUEST FROM A USER TO ACCESS THE DATABASE OBJECT
102
DETERMINE WHETHER AN ACCESS RESTRICTION IS IMPOSED ON THE DATABASE OBJECT
104
CONTROL ACCESS TO ELEMENTS IN THE DATABASE OBJECT BY THE USER BASED ON THE ACCESS RESTRICTION
106

FIG. 1

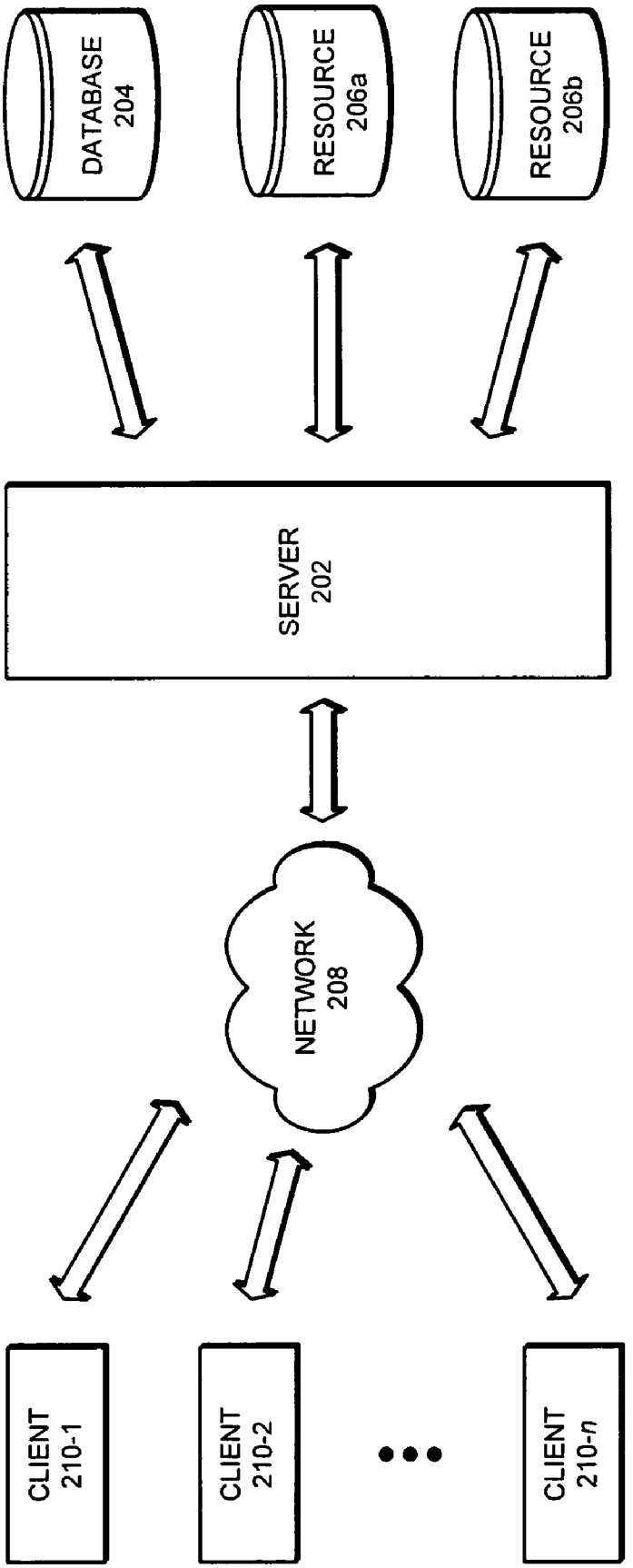

DATABASE 204
RESOURCE 206a
RESOURCE 206b
SERVER 202
NETWORK 208
CLIENT 210-1
CLIENT 210-2
CLIENT 210-n

FIG. 2

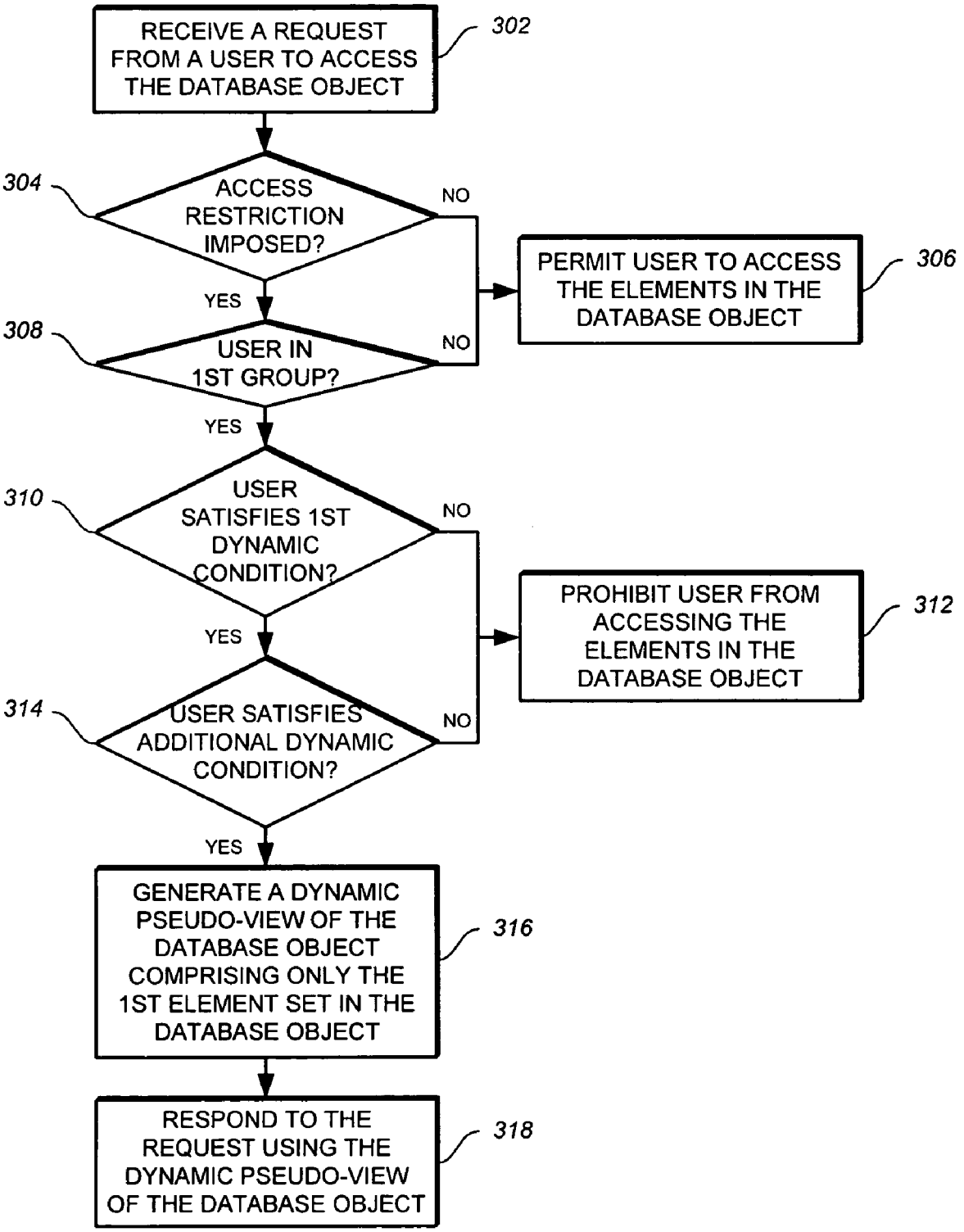

RECEIVE A REQUEST FROM A USER TO ACCESS THE DATABASE OBJECT
302
ACCESS RESTRICTION IMPOSED?
304
NO
YES
PERMIT USER TO ACCESS THE ELEMENTS IN THE DATABASE OBJECT
306
USER IN 1ST GROUP?
308
NO
YES
USER SATISFIES 1ST DYNAMIC CONDITION?
310
NO
YES
PROHIBIT USER FROM ACCESSING THE ELEMENTS IN THE DATABASE OBJECT
312
USER SATISFIES ADDITIONAL DYNAMIC CONDITION?
314
NO
YES
GENERATE A DYNAMIC PSEUDO-VIEW OF THE DATABASE OBJECT COMPRISING ONLY THE 1ST ELEMENT SET IN THE DATABASE OBJECT
316
RESPOND TO THE REQUEST USING THE DYNAMIC PSEUDO-VIEW OF THE DATABASE OBJECT
318

FIG. 3

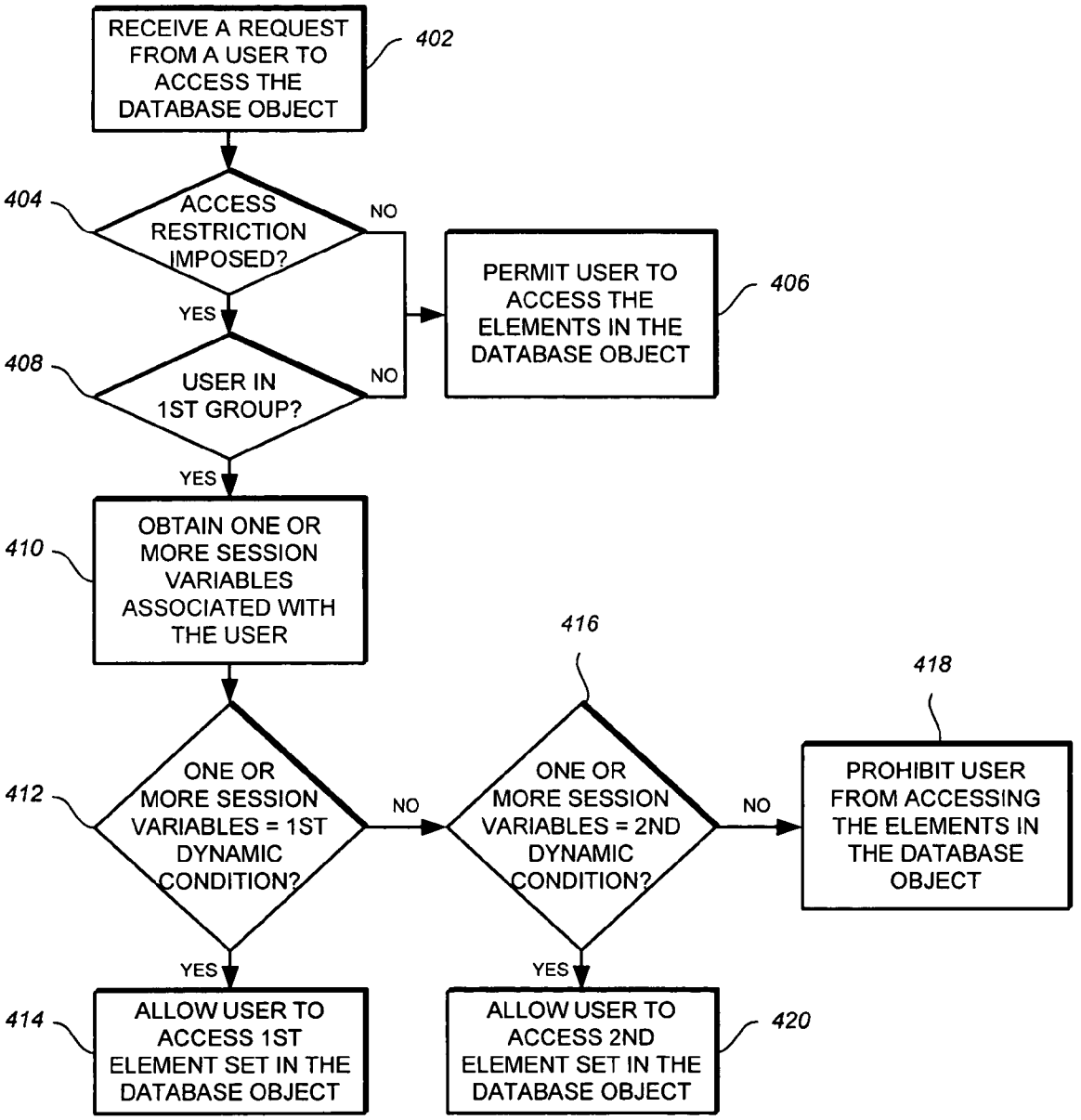

RECEIVE A REQUEST FROM A USER TO ACCESS THE DATABASE OBJECT
402
ACCESS RESTRICTION IMPOSED?
404
NO
YES
PERMIT USER TO ACCESS THE ELEMENTS IN THE DATABASE OBJECT
406
USER IN 1ST GROUP?
408
NO
YES
OBTAIN ONE OR MORE SESSION VARIABLES ASSOCIATED WITH THE USER
410
ONE OR MORE SESSION VARIABLES = 1ST DYNAMIC CONDITION?
412
NO
YES
ALLOW USER TO ACCESS 1ST ELEMENT SET IN THE DATABASE OBJECT
414
ONE OR MORE SESSION VARIABLES = 2ND DYNAMIC CONDITION?
416
NO
YES
ALLOW USER TO ACCESS 2ND ELEMENT SET IN THE DATABASE OBJECT
420
PROHIBIT USER FROM ACCESSING THE ELEMENTS IN THE DATABASE OBJECT
418

FIG. 4

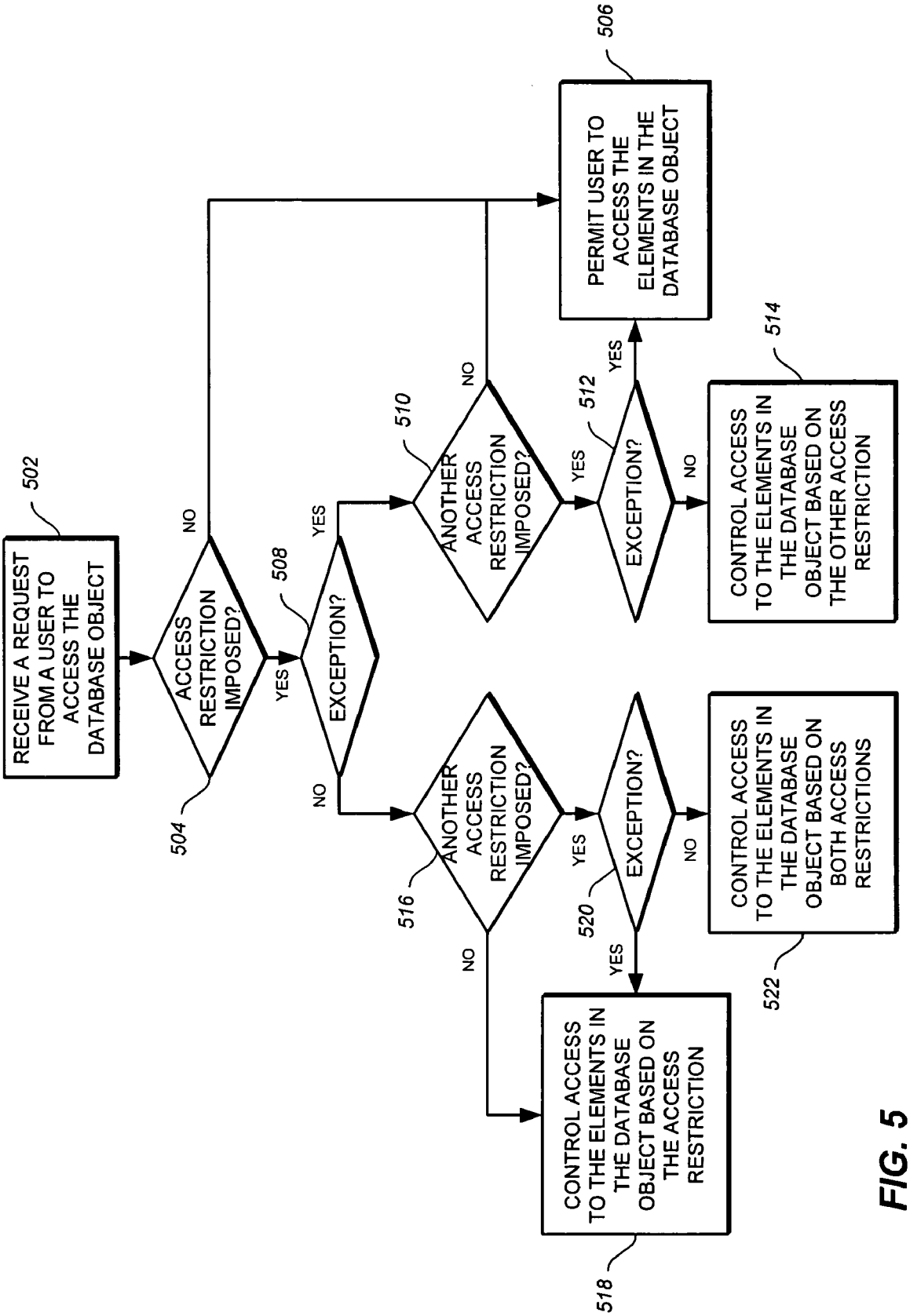

RECEIVE A REQUEST FROM A USER TO ACCESS THE DATABASE OBJECT
502
ACCESS RESTRICTION IMPOSED?
504
NO
YES
PERMIT USER TO ACCESS THE ELEMENTS IN THE DATABASE OBJECT
506
EXCEPTION?
508
YES
NO
ANOTHER ACCESS RESTRICTION IMPOSED?
510
NO
YES
EXCEPTION?
512
YES
NO
CONTROL ACCESS TO THE ELEMENTS IN THE DATABASE OBJECT BASED ON THE OTHER ACCESS RESTRICTION
514
ANOTHER ACCESS RESTRICTION IMPOSED?
516
NO
YES
EXCEPTION?
520
YES
NO
CONTROL ACCESS TO THE ELEMENTS IN THE DATABASE OBJECT BASED ON BOTH ACCESS RESTRICTIONS
522
CONTROL ACCESS TO THE ELEMENTS IN THE DATABASE OBJECT BASED ON THE ACCESS RESTRICTION
518

FIG. 5

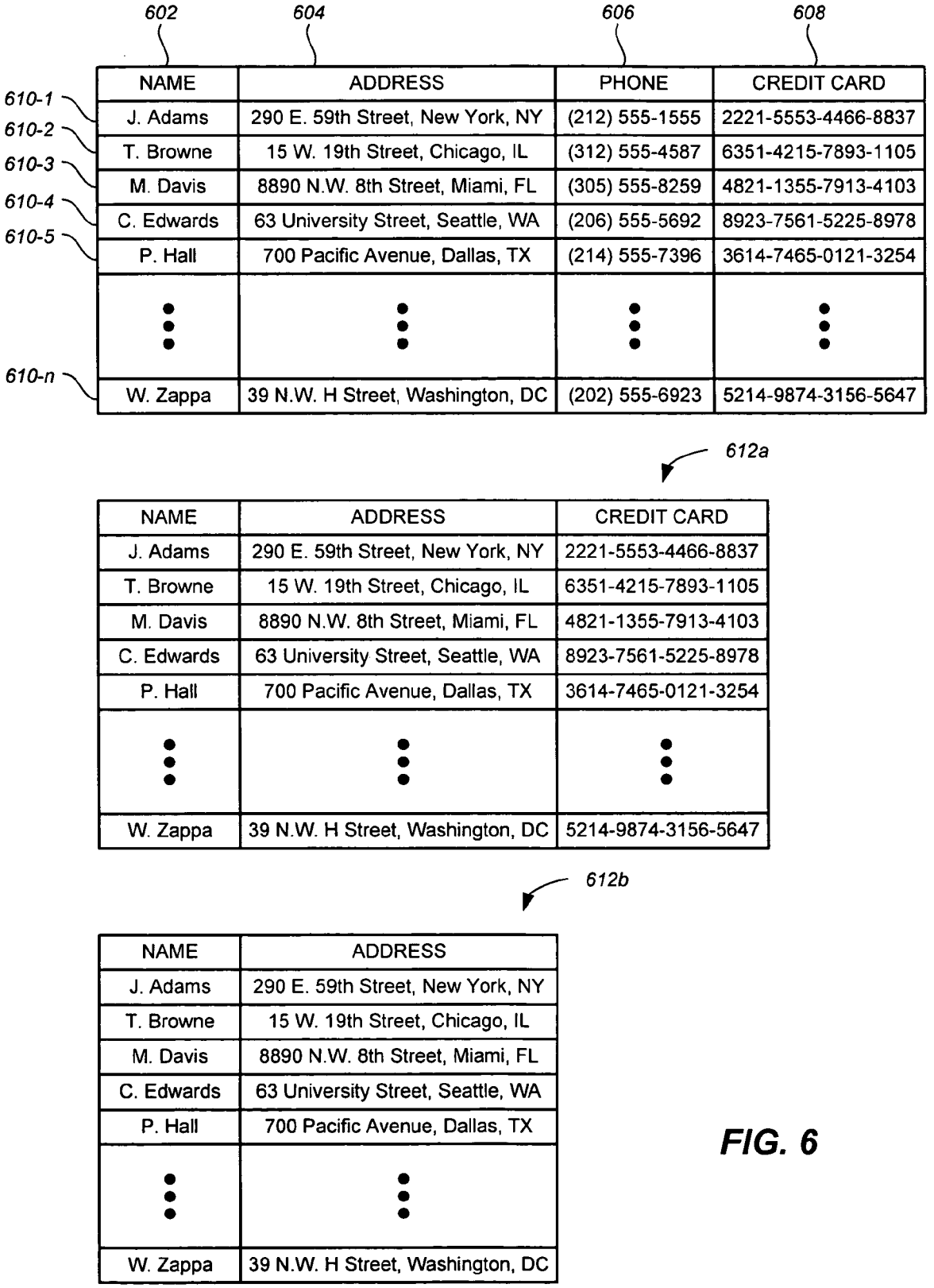

602
604
606
608
610-1
610-2
610-3
610-4
610-5
610-n
NAME
ADDRESS
PHONE
CREDIT CARD
J. Adams
290 E. 59th Street, New York, NY
(212) 555-1555
2221-5553-4466-8837
T. Browne
15 W. 19th Street, Chicago, IL
(312) 555-4587
6351-4215-7893-1105
M. Davis
8890 N.W. 8th Street, Miami, FL
(305) 555-8259
4821-1355-7913-4103
C. Edwards
63 University Street, Seattle, WA
(206) 555-5692
8923-7561-5225-8978
P. Hall
700 Pacific Avenue, Dallas, TX
(214) 555-7396
3614-7465-0121-3254
W. Zappa
39 N.W. H Street, Washington, DC
(202) 555-6923
5214-9874-3156-5647
612a
NAME
ADDRESS
CREDIT CARD
J. Adams
290 E. 59th Street, New York, NY
2221-5553-4466-8837
T. Browne
15 W. 19th Street, Chicago, IL
6351-4215-7893-1105
M. Davis
8890 N.W. 8th Street, Miami, FL
4821-1355-7913-4103
C. Edwards
63 University Street, Seattle, WA
8923-7561-5225-8978
P. Hall
700 Pacific Avenue, Dallas, TX
3614-7465-0121-3254
W. Zappa
39 N.W. H Street, Washington, DC
5214-9874-3156-5647
612b
NAME
ADDRESS
J. Adams
290 E. 59th Street, New York, NY
T. Browne
15 W. 19th Street, Chicago, IL
M. Davis
8890 N.W. 8th Street, Miami, FL
C. Edwards
63 University Street, Seattle, WA
P. Hall
700 Pacific Avenue, Dallas, TX
W. Zappa
39 N.W. H Street, Washington, DC

FIG. 6

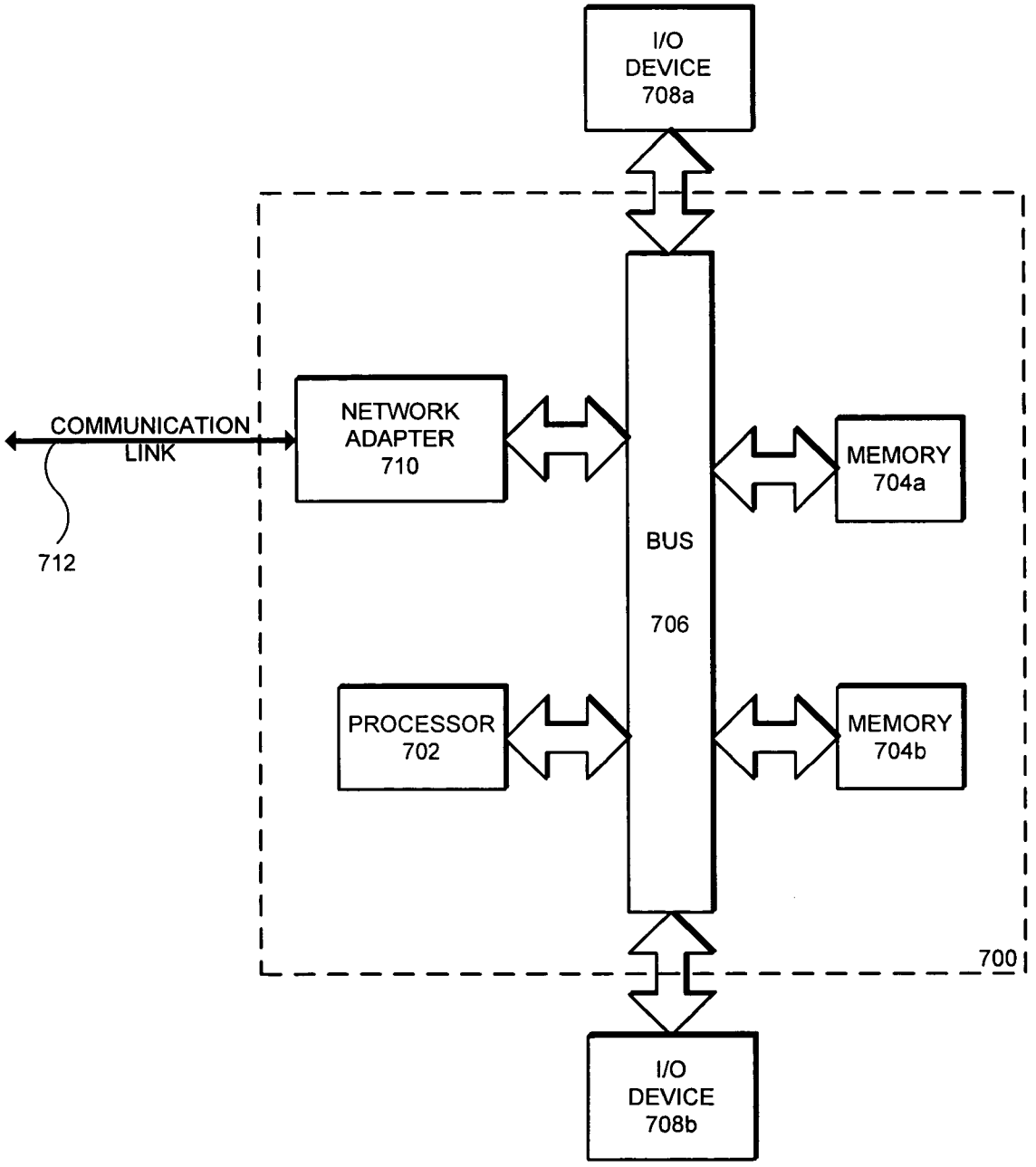

I/O
DEVICE
708a
COMMUNICATION
LINK
712
NETWORK
ADAPTER
710
BUS
706
MEMORY
704a
PROCESSOR
702
MEMORY
704b
700
I/O
DEVICE
708b

FIG. 7

ACCESS CONTROL FOR ELEMENTS IN A DATABASE OBJECT

FIELD OF THE INVENTION

The present invention relates generally to database management systems. More particularly, the present invention is directed to controlling access to elements in a database object.

BACKGROUND OF THE INVENTION

In a Database Management System (DBMS), data is stored in tables made up of records (e.g., rows) having one or more fields (e.g., columns). A view is a logical construct imposed over a table and is defined by metadata in the DBMS known as a view definition. The view definition contains mappings to one or more rows and columns in one or more tables stored in a database. Tables and views are considered to be database objects.

Fine-Grained Access Control (FGAC) is a mechanism by which the DBMS controls access to database object records and/or fields based on the identity of the user attempting to access the database object. FGAC complements the traditional Discretionary Access Control (DAC) implemented by many DBMS by allowing the DBMS to enforce two levels of access control: DAC is enforced at the object level (e.g., does the user have the right to access that table?) and FGAC is enforced at the element level (e.g., does the user have the right to access that row or column?).

Traditional methods of implementing FGAC within DBMS have relied upon the use of views. A view can be used to alter or restrict the data seen by a user using the view to access the underlying table(s). Views, however, have a number of shortcomings. For example, when the number of different restrictions is numerous, view definitions may become quite complex in an effort to incorporate all of the restrictions in one view, which strains system limits and makes maintenance of the view difficult.

Additionally, if a large number of simple views are desired, e.g., each one implementing a unique view of a table based on the restrictions for a specific set of users, the routing of user requests becomes difficult with the solution often being resolved within the database application rather than the DBMS. Furthermore, a user may be able to bypass the FGAC implemented through the views by accessing the base tables directly.

Another known implementation of FGAC is the use of user attributes to modify queries by adding predicates into the queries. A predicate is a condition that must be satisfied for the DBMS to return a value. In this approach, the user attributes (e.g., user identifier) are compared against a security policy defined within a procedure provided by the user on a table or view to make decisions regarding access to data. This approach allows row restrictions, traditionally handled by views, to be dynamically added to queries without requiring application modification.

One drawback of the query modification approach is that it only allows the DBMS to control access at the row-level. Views still have to be used to control access at the column-level. Additionally, the approach requires user programming of a strictly defined "predicate producing" procedure in order to implement a security policy. Moreover, query modification interferes with dynamic query caching because the modified queries will no longer match the original text of the queries, which makes query matching problematic and impacts the performance benefits of caching.

Further, the solutions described above fail to address the requirements from emerging privacy applications. Generally, a privacy policy indicates who can access what information, for what purpose, and resulting in what obligations. For example, a user John Doe may be allowed to access the credit card column from a customer table if he is using the billing application to process a customer order, but he may not be allowed to access that column for the purpose of sending marketing information to the customer. Existing FGAC solutions cannot address this requirement because they either do not support controlling access at the column level or they provide control access at the column level, but only for columns that have been statically defined (i.e., view-based techniques). Hence, a user is always restricted to a set of columns, regardless of the purpose for which he or she is accessing those columns.

Privacy applications are only one example where such flexibility is needed. Recent user requirements in the area of database security indicate that there is a need for database vendors to provide the notion of a session context. A session context is uniquely identified by a set of session attributes that may include the ID of the user who established that session, the IP address of the computer from which the user initiated the session, as well as other attributes as dictated by a particular implementation or scenario. Within a particular context, a user can have one or more privileges on one or more database objects that are not necessarily available to them within a different context. Thus, it is only natural that the next logical user requirement would be to allow certain columns to be accessible within one context, but not within another context. Currently, the only way to accomplish this would be to define a set of views that restrict access to certain columns and grant access on those views to users depending on their session context. Maintaining several views, however, has the same drawbacks mentioned earlier.

Accordingly, there is a need for a flexible mechanism to control access to elements in a database object based on one or more dynamic conditions, such as a session context or an access purpose without requiring the creation and maintenance of static views or the modification of queries. The present invention addresses such a need.

SUMMARY OF THE INVENTION

A method, computer program product, and system for controlling access to elements in a database object are provided. In this document, a group of one or more users is denoted as a user group and a set of one or more of the elements in a database object is denoted as an element set in the database object. The method, computer program product, and system provide for receiving a request from a user to access the database object, determining whether an access restriction is imposed on the database object, the access restriction specifying a first user group to which the access restriction is applicable, defining a first dynamic condition the first user group must satisfy in order to access the database object, and identifying a first element set in the database object accessible to the first user group when the first dynamic condition is satisfied, and controlling access to the elements in the database object by the user based on the access restriction.

Controlling access to elements in a database object using access restrictions, rather than views or modified queries, eliminates the worries concerning the creation and maintenance of complex views, the users bypassing restrictions by accessing underlying tables directly, the difficulties associated with routing user requests when there is a large number of views, the ability to control access at both the row and

column level, the need to program strictly defined "predicate producing" procedures, and the problems of dynamic query caching interferences. In addition, because the access restrictions are defined using one or more dynamic conditions, the flexibility needed to address current privacy and security concerns is achieved.

Particular implementations can include controlling access to the elements in the database object by confirming whether the user is in the first user group when the access restriction is imposed on the database object, verifying whether the user satisfies the first dynamic condition when the user is in the first user group, and allowing the user to access the first element set when the user satisfies the first dynamic condition.

Verifying whether the user satisfies the first dynamic condition may include obtaining one or more session variables associated with the user when the user is in the first user group and comparing the one or more session variable associated with the user to the first dynamic condition to determine whether the user satisfies the first dynamic condition. In an implementation, allowing the user to access the first element set in the database object comprises generating a dynamic pseudo-view of the database object comprising only the first element set in the database object when the user satisfies the first dynamic condition and responding to the request from the user using the dynamic pseudo-view of the database object.

In some embodiments, the database object is a table or a view, at least one element in the first element set is a column, the first dynamic condition is a session context or a session purpose associated with a user in the first user group, and the access restriction is stored in a database. Additionally, the access restriction can further define an additional dynamic condition the first user group must satisfy in order to access the first element set.

In other implementations, the access restriction further defines a second dynamic condition the first user group must alternatively satisfy in order to access the database object and further identifies a second element set in the database object accessible to the first user group when the second dynamic condition is satisfied. At least one element in the first element set may also be an element in the second element set.

Further aspects may include determining whether another access restriction is imposed on the database object, the other access restriction specifying a second user group to which the other access restriction is applicable. The other access restriction can also define another dynamic condition the second user group must satisfy in order to access the database object and identify another element set in the database object accessible to the second user group when the other dynamic condition is satisfied. In one embodiment, at least one user in the first user group is also a user in the second user group.

Another implementation also includes deciding whether an exception to the access restriction is applicable to the user requesting access to the database object and permitting the user to access the elements in the database object when the exception to the access restriction is applicable to the user.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a process flow of a method for controlling access to elements in a database object according to an aspect of the invention.

FIG. **2** illustrates a system according to one embodiment of the invention.

FIGS. **3-5** depict flowcharts of methods for controlling access to elements in a database object according to various implementations of the invention.

FIG. **6** shows a sample database object and exemplary dynamic pseudo-views generated based on the sample database object according to one aspect of the invention.

FIG. **7** is a block diagram of a data processing system with which embodiments of the present invention can be implemented.

DETAILED DESCRIPTION

The present invention relates generally to database systems and more particularly to controlling access to elements in a database object. The following description is presented to enable one of ordinary skill in the art to make and use the invention. Various modifications to the preferred implementations and the generic principles and features described herein will be readily apparent to those skilled in the art. Thus, the present invention is not intended to be limited to the implementations shown, but is to be accorded the widest scope consistent with the principles and features described herein.

FIG. **1** depicts a process **100** for controlling access to elements in a database object. At **102**, a request to access the database object is received from a user. A determination is then made at **104** as to whether an access restriction is imposed on the database object. In an embodiment, the access restriction specifies a first user group comprising one or more users to which the access restriction is applicable, defines a first dynamic condition the first user group must satisfy in order to access the database object, and identifies a first element set comprising one or more of the elements in the database object accessible to the first user group when the first dynamic condition is satisfied. In this document, a group of one or more users is denoted as a user group and a set of one or more of the elements in a database object is denoted as an element set in the database object.

Access to the elements in the database object by the user is controlled based on the access restriction when the access restriction is imposed on the database object and the user is in the first user group (**106**). In some implementations, the database object is a table or a view, at least one element in the first element set is a column, and the first dynamic condition is a session context or a session purpose associated with a user in the first user group.

A session purpose could be determined based on the type of application the user is employing when requesting access to the database object, for example, a purchasing application or a marketing application. A session context could be the location from which the user is requesting access to the database object, for instance, from the office or at home. The location may be determined based on the IP address of the computer from which the user is requesting access.

Session context and session purpose are just two examples of dynamic conditions. A dynamic condition can also be a function. For example, the condition can be "F(current time) is TRUE" where "F" is a function that compares the current time to the time of the day when access can be granted.

Illustrated in FIG. **2** is a system **200** including a server **202** interconnected to clients **210-1** to **210-*n*** via a network **208**. Server **202** and clients **210-1** to **210-*n*** may be any data processing system, such as computers, workstations, and handheld portable devices. In addition, system **200** may include more or less clients in other embodiments. Network **208** may be the Internet or World Wide Web (WWW) in some implementations.

System **200** also includes a database **204** and resources **206***a*-**206***b*. Each resource may be a storage media, a database, a set of XML (eXtensible Markup Language) documents, a directory service, such as LDAP (Lightweight Directory Access Protocol) server, or a backend system. Other embodiments of system **200** may include more or less databases and/or resources.

Database **204** and resources **206***a*-**206***b* are coupled to server **202**. The interface between server **202** and database **204** and resources **206***a*-**206***b* may be a local area network, Internet, a proprietary interface, or any combination of the foregoing. Clients **210-1** to **210**-*n* can access database **204** and resources **206***a*-**206***b* through server **202**. Any of server **202**, database **204**, resources **206***a*-**206***b*, and clients **210-1** to **210**-*n* may be located remotely from one another or may share a location.

The configuration of system **200** is not intended as a limitation of the present invention, as will be understood by those of ordinary skill in the art from a review of the following detailed description. For example, network **208** may comprise a wireless link, a telephone communication, a radio communication, or a computer network (e.g., a Local Area Network (LAN) or a Wide Area Network (WAN)).

In one implementation, database **204** is operable to store a database object comprising a plurality of elements and server **202** is operable to receive a request from a user to access the database object. The request may be submitted by the user through one of clients **210-1** to **210**-*n*. Server **202** is also operable to determine whether an access restriction is imposed on the database object. The access restriction specifies a first user group to which the access restriction is applicable, defines a first dynamic condition the first user group must satisfy in order to access the database object, and identifies a first element set in the database object accessible to the first user group when the first dynamic condition is satisfied.

Server **202** is then operable to control access to the elements in the database object by the user based on the access restriction when the access restriction is imposed on the database object and the user is in the first user group. In some embodiments, database **204** is further operable to store the access restriction. The access restriction may be stored in a catalog of database **204** (not shown).

FIG. **3** shows a process **300** for controlling access to elements in the database object according to an aspect of the invention. A request to access the database object is received from a user at **302**. At **304**, a determination is made as to whether an access restriction is imposed on the database object. The access restriction specifies a first user group to which the access restriction is applicable, defines a first dynamic condition and an additional dynamic condition the first user group must satisfy in order to access the database object, and identifies a first element set in the database object accessible to the first user group when the first dynamic condition and the additional dynamic condition are satisfied.

If no access restriction is imposed on the database object, the user is permitted to access the elements in the database object (**306**). However, if the access restriction is imposed on the database object, process **300** confirms whether the user is in the first user group to which the access restriction is applicable (**308**). When the user is not in the first user group, process **300** proceeds to **306** and the user is permitted to access the elements in the database object.

When the user is in the first user group, process **300** verifies whether the user satisfies the first dynamic condition (**310**). If the user does not satisfy the first dynamic condition, the user is prohibited from accessing the elements in the database object (**312**). If the user does satisfy the first dynamic condition, process **300** verifies whether the user satisfies the additional dynamic condition (**314**). When the user fails to satisfy the additional dynamic condition, process **300** proceeds to **312** and prohibits the user from accessing the elements in the database object.

A dynamic pseudo-view of the database object comprising only the first element set is generated when the user satisfies the first dynamic condition and the additional dynamic condition (**316**). The request from the user is then responded to using the dynamic pseudo-view of the database object (**318**). A dynamic pseudo-view is a view-like entity with attributes similar to a predefined regular view. However, because it is dynamically created, it does not exist in a database, such as database **204** in FIG. **2**, and has no dependencies.

Depicted in FIG. **4** is another process **400** for controlling access to elements in a database object. At **402**, a request is received from a user to access the database object. A determination is then made at **404** as to whether an access restriction is imposed on the database object. The access restriction specifies a first user group to which the access restriction is applicable, defines a first dynamic condition the first user group must satisfy in order to access the database object, and identifies a first element set in the database object accessible to the first user group when the first dynamic condition is satisfied.

In the embodiment, the access restriction also defines a second dynamic condition the first user group must alternatively satisfy in order to access the database object and identifies a second element set in the database object accessible to the first user group when the second dynamic condition is satisfied. In some implementations, at least one element in the first element set is also an element in the second element set.

When no access restrictions are imposed on the database object, the user is permitted to access the elements in the database object (**406**). When the access restriction is imposed on the database object, process **400** confirms whether the user is in the first user group (**408**). If the user is not in the first user group, process **400** proceeds to **406** and permits the user to access the elements in the database object.

If the user is in the first user group, one or more session variables associated with the user is obtained (**410**). In one embodiment, when the user establishes a session through some application, a session start trigger will populate one or more session variables associated with the user with the appropriate values based on information from the user and the application. The session start trigger is a program that is automatically executed when a session is established. Process **400** then compares the one or more session variables associated with the user to the first dynamic condition to determine whether the user satisfies the first dynamic condition (**412**).

The user is allowed to access the first element set in the database object when the user satisfies the first dynamic condition, i.e., the one or more session variables match or correspond to the first dynamic condition (**414**). When the one or more session variables do not match the first dynamic condition, process **400** compares them to the second dynamic condition (**416**). If they also fail to match the second dynamic condition, the user is prohibited from accessing the elements in the database object (**418**). However, if the one or more session variables associated with the user match the second dynamic condition, the user is allowed to access the second element set in the database object (**420**).

FIG. **5** illustrates a process **500** for controlling access to elements in a database object according to a further embodiment of the invention. A request to access the database object is received from a user at **502**. A determination is then made at **504** as to whether an access restriction has been imposed on

the database object. The access restriction specifies a first user group to which the access restriction is applicable, defines a first dynamic condition the first user group must satisfy in order to access the database object, and identifies a first element set in the database object accessible to the first user group when the first dynamic condition is satisfied.

If no access restrictions are imposed on the database object, the user is permitted to access the elements in the database object (**506**). If, however, the access restriction has been imposed on the database object, process **500** decides whether an exception to the access restriction is applicable to the user requesting access to the database object (**508**). When the exception to the access restriction is applicable to the user at block **508**, a determination is made as to whether another access restriction is imposed on the database object, the other access restriction specifies a second user group to which the other access restriction is applicable (**510**). In an implementation, at least one user in the first user group is also a user in the second user group.

The other access restriction may further define another dynamic condition the second user group must satisfy in order to access the database object and identify another element set in the database object accessible to the second user group when the other dynamic condition is satisfied. Additionally, the other element set in the database object may be a subset of the first element set.

Process **500** will proceed to **506** to permit the user to access the elements in the database object when no other access restrictions are imposed on the database object. However, it will decide whether an exception to the other access restriction is applicable to the user requesting access to the database object when the other access restriction is also imposed on the database object (**512**). The user is permitted to access the elements in the database object if the exception to the other access restriction is applicable to the user (**506**). In contrast, access to the elements in the database object by the user is controlled based on the other access restriction if the exception to the other access restriction is inapplicable to the user (**514**).

When the exception to the access restriction is not applicable to the user at block **508**, a determination is made as to whether another access restriction is imposed on the database object (**516**). If no other access restrictions are imposed on the database object, access to the elements in the database object by the user is controlled based on the access restriction (**518**). However, if another access restriction is imposed on the database object, process **500** will decide whether an exception to the other access restriction is applicable to the user requesting access to the database object (**520**).

Access to the elements in the database object by the user will be controlled based on the access restriction when the exception to the other access restriction is applicable to the user (**518**). Conversely, access to the elements in the database object by the user will be controlled based on both access restrictions when the exception to the other access restriction is not applicable to the user (**522**).

Shown in FIG. **6** is a sample database object **600** with elements **602-610**. Database object **600** is a table called "customer data" with a column **602** for names, a column **604** for addresses, a column **606** for phone numbers, and a column **608** for credit card numbers. Table **600** has n number of rows **610-1** to **610-*n***. Embodiments of the present invention enables access restrictions to be created such that it becomes possible to express which elements **602-610** in database object **600** are accessible by a user and under what condition.

For example, suppose a user named "Bob" is allowed to access columns **602**, **604**, and **608** in table **600** for the purpose of "Billing" and only columns **602** and **604** for the purpose of "Marketing." The following Structured Query Language (SQL) statement illustrates how an access restriction can be created to limit user Bob's access to columns **602-610** in table **600** based on the purpose of access.

```
CREATE RESTRICTION r1
ON TABLE customer data
FOR Bob
TO COLUMNS
  (name, address, credit card) WHEN (SessionVariablePurpose ='Billing')
  (name, address) WHEN (SessionVariablePurpose ='Marketing')
```

Thus, when table **600** is queried by user Bob, server **202** in FIG. **2** for example, can determine that an access restriction applies for user Bob. Server **202** may then look up a session variable "SessionVariablePurpose" associated with user Bob and read its value. If it is set to "Billing," server **202** will implement access restriction "**r1**" in the query plan as if that restriction was statically defined as follows:

```
CREATE RESTRICTION r1
ON TABLE customer data
FOR Bob
TO COLUMNS (name, address, credit card)
```

A dynamic pseudo-view **612***a* of table **600** that is depicted in FIG. **6** can be generated to respond to user Bob's queries to table **600**.

However, if the value of the session variable "SessionVariablePurpose" was "Marketing," then server **202** will implement restriction "**r1**" in the query plan as if that restriction was statically defined as follows:

```
CREATE RESTRICTION r1
ON TABLE customer data
FOR Bob
TO COLUMNS (name, address)
```

A dynamic pseudo-view **612***b* of table **600**, which is illustrated in FIG. **6**, will be generated to respond to user Bob's queries on table **600**. For more information regarding the creation and use of access restrictions, see "A Method for Implementing Fine-Grained Access Control Using Access Restrictions," U.S. patent application Ser. No. 10/837,387, filed on Apr. 30, 2004, which is hereby incorporated by reference in its entirety for all purposes.

The invention can take the form of an entirely hardware embodiment, an entirely software embodiment, or an embodiment containing both hardware and software elements. In one aspect, the invention is implemented in software, which includes, but is not limited to, firmware, resident software, microcode, etc.

Furthermore, the invention can take the form of a computer program product accessible from a computer-usable or computer-readable medium providing program code for use by or in connection with a computer or any instruction execution system. For the purposes of this description, a computer-usable or computer-readable medium can be any apparatus that can contain, store, communicate, propagate, or transport the program for use by or in connection with the instruction execution system, apparatus, or device.

The medium can be an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system (or apparatus or device) or a propagation medium. Examples of a computer-readable medium include a semiconductor or solid state memory, magnetic tape, a removable computer diskette, a random access memory (RAM), a read-only memory (ROM), a rigid magnetic disk, and an optical disk. Current examples

of optical disks include DVD, compact disk-read-only memory (CD-ROM), and compact disk-read/write (CD-R/W).

FIG. **7** depicts a data processing system **700** suitable for storing and/or executing program code. Data processing system **700** includes a processor **702** coupled to memory elements **704***a-b* through a system bus **706**. In other embodiments, data processing system **700** may include more than one processor and each processor may be coupled directly or indirectly to one or more memory elements through a system bus.

Memory elements **704***a-b* can include local memory employed during actual execution of the program code, bulk storage, and cache memories that provide temporary storage of at least some program code in order to reduce the number of times the code must be retrieved from bulk storage during execution. As shown, input/output or I/O devices **708***a-b* (including, but not limited to, keyboards, displays, pointing devices, etc.) are coupled to data processing system **700**. I/O devices **708***a-b* may be coupled to data processing system **700** directly or indirectly through intervening I/O controllers (not shown).

In the embodiment, a network adapter **710** is coupled to data processing system **700** to enable data processing system **700** to become coupled to other data processing systems or remote printers or storage devices through communication link **712**. Communication link **712** can be a private or public network. Modems, cable modems, and Ethernet cards are just a few of the currently available types of network adapters.

Various implementations for controlling access to elements in a database object have been described. Nevertheless, one of ordinary skill in the art will readily recognize that various modifications may be made to the implementations, and any variations would be within the spirit and scope of the present invention. For example, the above-described process flows are described with reference to a particular ordering of process actions. However, the ordering of many of the described process actions may be changed without affecting the scope or operation of the invention. Accordingly, many modifications may be made by one of ordinary skill in the art without departing from the spirit and scope of the following claims.

What is claimed is:

1. A method of controlling access to elements in a database object, the method comprising:

receiving a request from a user to access the database object, wherein the request includes a query to retrieve information from the database object;

determining whether an access restriction is imposed on the database object, the access restriction specifying a first user group to which the access restriction is applicable, defining a first dynamic condition the first user group must satisfy in order to access the database object, and identifying a first element set in the database object accessible to the first user group when the first dynamic condition is satisfied, wherein the first element set includes at least one, and less than all, table columns of the database object to restrict access to one or more table columns, wherein the first dynamic condition indicates access information including one or more of a session context and session purpose for the user to access the database object, and wherein two or more of said session contexts and purposes for the user to access the database object enable access to be restricted to at least one different table column of said database object; and

controlling access to the elements in the database object by the user based on the access restriction, wherein controlling access to the elements in the database objects comprises:

confirming whether the user is in the first user group when the access restriction is imposed on the database object;

verifying whether the user satisfies the first dynamic condition when the user is in the first user group by ascertaining session information for the user from one or more session variables associated with the user, wherein the session information includes one or more of the session context and session purpose for access of the database object, and comparing the session information for the user against the access information indicated by the first dynamic condition to determine satisfaction of that condition; and

allowing the user to access the first element set when the user satisfies the first dynamic condition, wherein allowing the user to access the first element set comprises:

dynamically generating a dynamic pseudo-view of the database object comprising only the first element set in response to said verification of the user satisfying the first dynamic condition; and

responding to the request from the user by applying the received query to the dynamic pseudo-view of the database object to retrieve the information.

2. The method of claim **1**, wherein the database object is a table or a view.

3. The method of claim **1**, wherein the access restriction further defines a second dynamic condition the first user group must alternatively satisfy in order to access the database object and further identifies a second element set in the database object accessible to the first user group when the second dynamic condition is satisfied.

4. The method of claim **3**, wherein at least one element in the first element set is also an element in the second element set.

5. The method of claim **1**, wherein the access restriction further defines an additional dynamic condition the first user group must satisfy in order to access the first element set.

6. The method of claim **1**, further comprising:

determining whether another access restriction is imposed on the database object, the other access restriction specifying a second user group to which the other access restriction is applicable.

7. The method of claim **6**, wherein the other access restriction further defines another dynamic condition the second user group must satisfy in order to access the database object and identifies another element set in the database object accessible to the second user group when the other dynamic condition is satisfied.

8. The method of claim **7**, wherein the other element set is a subset of the first element set.

9. The method of claim **6**, wherein at least one user in the first user group is also a user in the second user group.

10. The method of claim **1**, further comprising:

deciding whether an exception to the access restriction is applicable to the user requesting access to the database object; and

permitting the user to access the elements in the database object when the exception to the access restriction is applicable to the user.

11. A system comprising:

a database operable to store a database object, the database object comprising elements; and

a server coupled to the database, the server comprising a processor and a memory, the server being operable to:

receive a request from a user to access the database object, wherein the request includes a query to retrieve information from the database object;

determine whether an access restriction is imposed on the database object, the access restriction specifying a first user group to which the access restriction is applicable, defining a first dynamic condition the first user group must satisfy in order to access the database object, and identifying a first element set in the database object accessible to the first user group when the first dynamic condition is satisfied, wherein the first element set includes at least one, and less than all, table columns of the database object to restrict access to one or more table columns, wherein the first dynamic condition indicates access information including one or more of a session context and session purpose for the user to access the database object, and wherein two or more of said session contexts and purposes for the user to access the database object enable access to be restricted to at least one different table column of said database object; and

control access to the elements in the database object by the user based on the access restriction, wherein controlling access to the elements in the database object comprises:

confirming whether the user is in the first user group when the access restriction is imposed on the database object;

verifying whether the user satisfies the first dynamic condition when the user is in the first user group by ascertaining session information for the user from one or more session variables associated with the user, wherein the session information includes one or more of the session context and session purpose for access of the database object, and comparing the session information for the user against the access information indicated by the first dynamic condition to determine satisfaction of that condition; and

allowing the user to access the first element set when the user satisfies the first dynamic condition, wherein allowing the user to access the first element set comprises:

dynamically generating a dynamic pseudo-view of the database object comprising only the first element set in response to said verification of the user satisfying the first dynamic condition; and

responding to the request from the user by applying the received query to the dynamic pseudo-view of the database object to retrieve the information.

12. The system of claim **11**, wherein the database object is a table or a view.

13. The system of claim **11**, wherein the access restriction further defines a second dynamic condition the first user group must alternatively satisfy in order to access the database object and further identifies a second element set in the database object accessible to the first user group when the second dynamic condition is satisfied.

14. The system of claim **13**, wherein at least one element in the first element set is also an element in the second element set.

15. The system of claim **11**, wherein the access restriction further defines an additional dynamic condition the first user group must satisfy in order to access the first element set.

16. The system of claim **11**, wherein the server is further operable to:

determine whether another access restriction is imposed on the database object, the other access restriction specifying a second user group to which the other access restriction is applicable.

17. The system of claim **16**, wherein the other access restriction further defines another dynamic condition the second user group must satisfy in order to access the database object and identifies another element set in the database object accessible to the second user group when the other dynamic condition is satisfied.

18. The system of claim **17**, wherein the other element set is a subset of the first element set.

19. The system of claim **16**, wherein at least one user in the first user group is also a user in the second user group.

20. The system of claim **11**, wherein the server is further operable to:

decide whether an exception to the access restriction is applicable to the user requesting access to the database object; and

permit the user to access the elements in the database object when the exception to the access restriction is applicable to the user.

21. A computer program product comprising a computer-readable storage medium, the computer-readable storage medium including a computer-readable program for controlling access to elements in a database object, wherein the computer-readable program when executed on a computer causes the computer to:

receive a request from a user to access the database object, wherein the request includes a query to retrieve information from the database object;

determine whether an access restriction is imposed on the database object, the access restriction specifying a first user group to which the access restriction is applicable, defining a first dynamic condition the first user group must satisfy in order to access the database object, and identifying a first element set in the database object accessible to the first user group when the first dynamic condition is satisfied, wherein the first element set includes at least one, and less than all, table columns of the database object to restrict access to one or more table columns, wherein the first dynamic condition indicates access information including one or more of a session context and session purpose for the user to access the database object, and wherein two or more of said session contexts and purposes for the user to access the database object enable access to be restricted to at least one different table column of said database object; and

control access to the elements in the database object by the user based on the access restriction, wherein controlling access to the elements in the database object comprises:

confirming whether the user is in the first user group when the access restriction is imposed on the database object;

verifying whether the user satisfies the first dynamic condition when the user is in the first user group by ascertaining session information for the user from one or more session variables associated with the user, wherein the session information includes one or more of the session context and session purpose for access of the database object, and comparing the session information for the user against the access information indicated by the first dynamic condition to determine satisfaction of that condition; and

allowing the user to access the first element set when the user satisfies the first dynamic condition, wherein allowing the user to access the first element set comprises:

dynamically generating a dynamic pseudo-view of the database object comprising only the first element set in response to said verification of the user satisfying the first dynamic condition; and

responding to the request from the user by applying the received query to the dynamic pseudo-view of the database object to retrieve the information.

22. The computer program product of claim **21**, wherein the database object is a table or a view.

23. The computer program product of claim **21**, wherein the access restriction further defines a second dynamic condition the first user group must alternatively satisfy in order to access the database object and further identifies a second element set in the database object accessible to the first user group when the second dynamic condition is satisfied.

24. The computer program product of claim **23**, wherein at least one element in the first element set is also an element in the second element set.

25. The computer program product of claim **21**, wherein the access restriction further defines an additional dynamic condition the first user group must satisfy in order to access the first element set.

26. The computer program product of claim **21**, wherein the computer-readable program when executed on the computer further causes the computer to:

determine whether another access restriction is imposed on the database object, the other access restriction specifying a second user group to which the other access restriction is applicable.

27. The computer program product of claim **26**, wherein the other access restriction further defines another dynamic condition the second user group must satisfy in order to access the database object and identifies another element set in the database object accessible to the second user group when the other dynamic condition is satisfied.

28. The computer program product of claim **27**, wherein the other element set is a subset of the first element set.

29. The computer program product of claim **26**, wherein at least one user in the first user group is also a user in the second user group.

30. The computer program product of claim **21**, wherein the computer-readable program when executed on the computer further causes the computer to:

decide whether an exception to the access restriction is applicable to the user requesting access to the database object; and

permit the user to access the elements in the database object when the exception to the access restriction is applicable to the user.

* * * * *

US007243097B1

(12) **United States Patent**
Agrawal et al.

(10) **Patent No.: US 7,243,097 B1**
(45) **Date of Patent: Jul. 10, 2007**

(54) **EXTENDING RELATIONAL DATABASE SYSTEMS TO AUTOMATICALLY ENFORCE PRIVACY POLICIES**

(75) Inventors: **Rakesh Agrawal**, San Jose, CA (US); **Paul Miller Bird**, Markham (CA); **Tyrone W. A. Grandison**, San Jose, CA (US); **Gerald George Kiernan**, San Jose, CA (US); **Scott Ian Logan**, Don Mills (CA); **Walid Rjaibi**, Markham (CA)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **11/307,763**

(22) Filed: **Feb. 21, 2006**

(51) **Int. Cl.**
G06F 7/00 (2006.01)
G06F 17/00 (2006.01)

(52) **U.S. Cl.** **707/3**; 707/100

(58) **Field of Classification Search** 707/3, 707/9, 10, 100; 709/220, 225
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,941,947	A *	8/1999	Brown et al.	709/225
6,065,012	A	5/2000	Balsara et al.	
6,253,203	B1 *	6/2001	O'Flaherty et al.	707/9
6,496,832	B2	12/2002	Chi et al.	
2002/0095405	A1 *	7/2002	Fujiwara	707/3
2004/0215626	A1	10/2004	Colossi et al.	
2005/0144176	A1 *	6/2005	Lei et al.	707/100
2005/0289342	A1	12/2005	Needham	

OTHER PUBLICATIONS

P3P Definition, obtained from www.wikipedia.org, Jun. 27, 2006.*

Graefe, Goetz; "Query Evaluation Techniques for Large Databases"; ACM Computing Surveys, vol. 25, No. 2, Jun. 1993.

Agrawal et al.; "Hippocratic Databases"; Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002.

LeFevre et al.; "Limiting Disclosure in Hippocratic Databases"; Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004.

"New Features in Adaptive Server Version 12.5"; What's New in Adaptive Server Enterprise?; Chapter 1; http://manuals.sybase.com/onlinebooks/group-as/asg1250e/whatsnew/@Generic_BookTextView/445;hf=0#X.

"Fine Grained Access Control and Application Contexts" Ask Tom (part of Oracle Magazine); Jun. 1999; http://asktom.oracle.com/~tkyte/article2/index.html.

Stonebraker et al.; "Access Control in a Relational Data Base Management System by Query Modification"; Department of Electrical Engineering and Computer Sciences and the Electrical Research Laboratory, University of California, Berkley, California 94720.

Chamberlin, Don; "A Complete Guide to DB2 Universal Database"; pp. 122-128; Morgan Kaufmann Publ.; Jun. 1998.

Nanda et al; "Oracle Privacy Security Auditing: Includes Federal Law Compliance with HIPAA, Sarbanes-Oxley & The Gramm-Leach-Bliley Act GLB"; pp. 240-251; Oracles in Focus; Rampant TechPress; Dec. 2003.

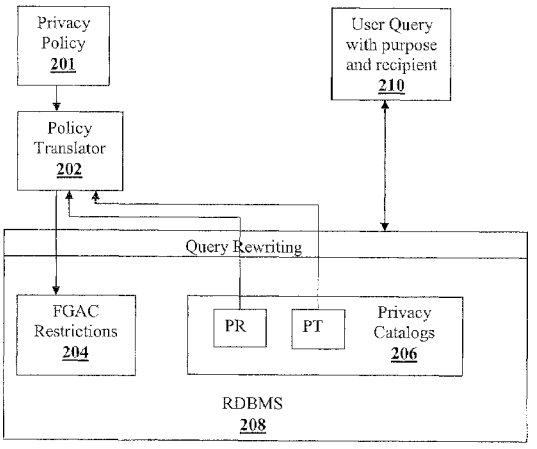
* cited by examiner

Primary Examiner—Sam Rimell
(74) *Attorney, Agent, or Firm*—Ramraj Soundararajan; IP Authority, LLC

(57) **ABSTRACT**

A method of transforming relational database management systems into their privacy-preserving equivalents is provided. Language constructs allow fine grained access control (FGAC) restrictions to be specified on the access to data in a table at the level of a row, a column or a cell. Fine grained restrictions are a combination of access control and privacy policy restrictions, which ensure compliance with current privacy legislation mandates.

13 Claims, 6 Drawing Sheets



```
create restriction restriction-name
on table-x
for auth-name-1 [ except auth-name-2]
( (    (to columns column-name-list)
     | (to rows [ where search-condition ] )
     | (to cells (column-name-list [ where search-condition ] )+ )
  )
  [ for purpose purpose-list ]
  [ for recipient recipient-list ]
)+
command-restriction
```

Figure 1

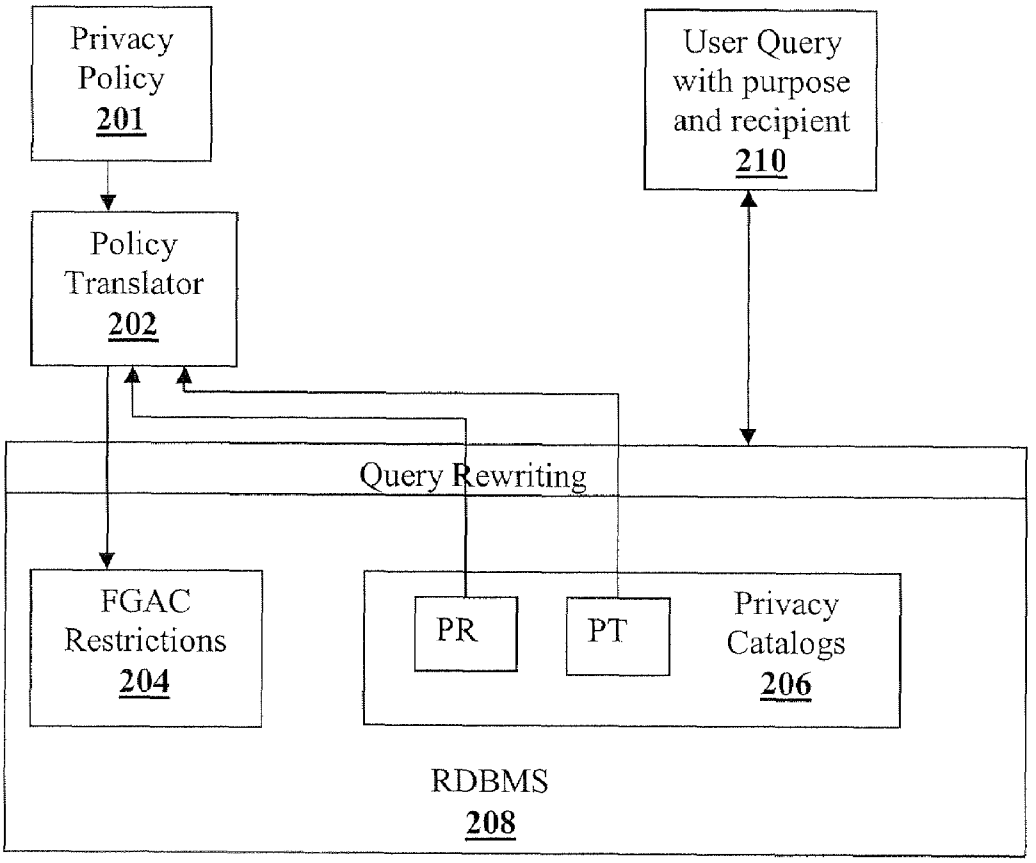


Figure 2

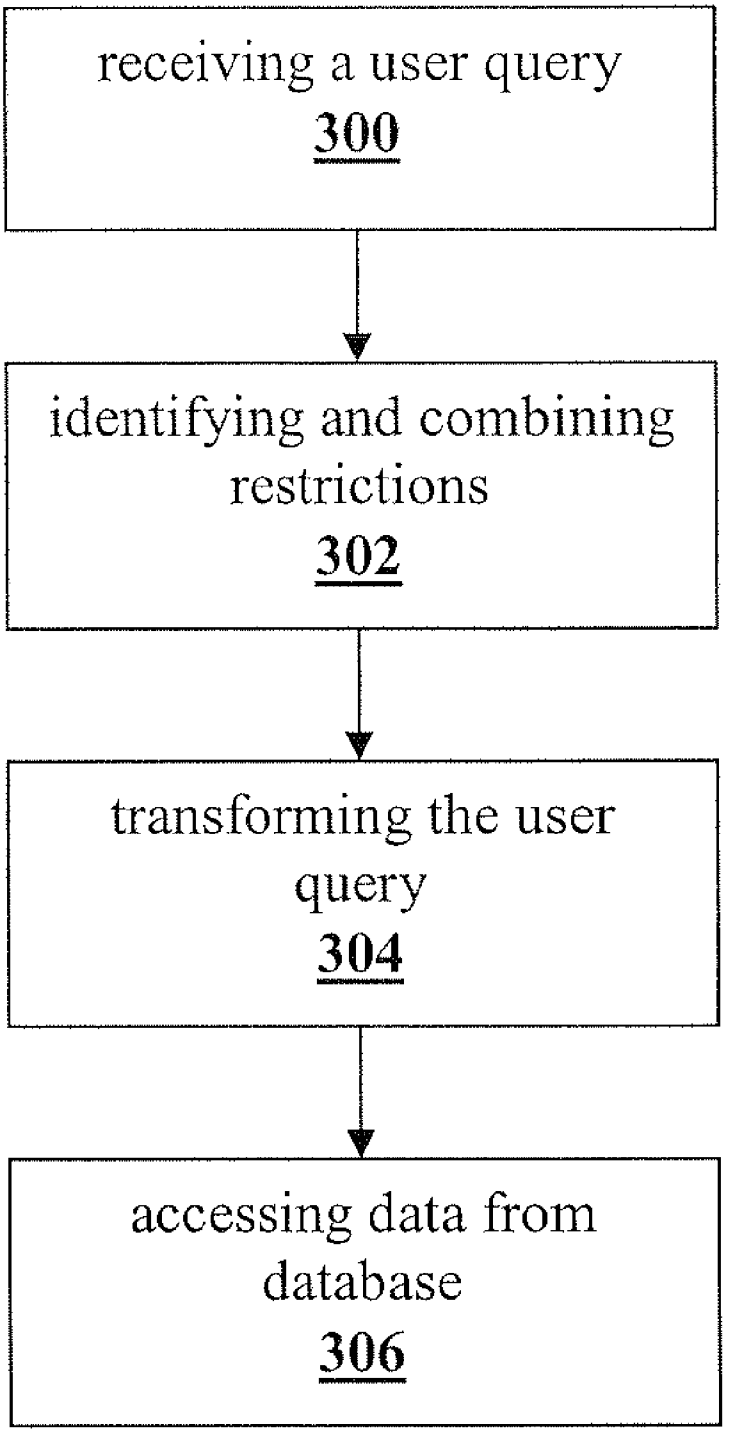

receiving a user query
300
identifying and combining restrictions
302
transforming the user query
304
accessing data from database
306

Figure 3

```
1   for each table reference t in query Q do begin
2     if (exists a restriction r pertaining to t for Q) then begin
3       create a dynamic view v ∈ Q over t
4       replace each reference to t ∈ Q with a reference to v ∈ Q

        // create the dynamic view v using
        // the following print statements
        //
5       print "select"
6       for each column c ∈ t do begin
          // c_p, c_r are the purposes, recipients
          // of column c in restriction r
          // Q_p, Q_r are the purpose, recipient of query Q
          //
7         if (c ∉ r|Q_p ∈ c_p ∧ Q_r ∈ c_r
            // c isn't included in the restriction r
            // access to c is thus prohibited
            //
8           print "null"
9         else begin
            // The whereClause function returns
            // the predicate associated with c
            // that is specified in the restriction
            //
10          let w = whereClause(c)
11          if w = null then
              // There is no "where" condition
              // governing the use of c ∈ r, thus access
              // to all column values is granted unconditionally
              //
12            print c.colname
13          else begin
              // Implement the "where" condition
              // using a SQL case statement to grant
              // only conditional access to the column c
              //
14            print "case when exists ("
15            print w.condition
16            print ")"
17            print "then"
18            print c.colname
19            print "else null end as"
20            print c.colname
21          end
22        end
23      end
24      print "from"
25      print t.tablename
26    end
```

Figure 4

```
...
<!-- Statement1 -->
<STATEMENT>
 <CONSEQUENCE>
  Encodes that personal and medical information
  can be accessed for emergency purposes
  by ourselves
 </CONSEQUENCE>
 <PURPOSE>
  <other-purpose>
    Emergency
  </other-purpose>
 </PURPOSE>
 <RECIPIENT><ours/></RECIPIENT>
 <RETENTION><stated-purpose/></RETENTION>
 <DATA-GROUP>
  <DATA ref = "#personal"/>
  <DATA ref = "#medical">
   <CATEGORIES>
    <health/>
   </CATEGORIES>
  </DATA>
 </DATA-GROUP>
</STATEMENT>

<!-- Statement2 -->
 <STATEMENT>
  <CONSEQUENCE>
   Encodes that we and drug companies
   with the same data usage policies
   can access personal and medical information
   for new_drug_research on an opt-out basis
  </CONSEQUENCE>
  <PURPOSE><develop/></PURPOSE>
  <RECIPIENT>
   <ours required="opt-out"/>
   <same required="opt-out"/>
  </RECIPIENT>
  <RETENTION><stated-purpose/></RETENTION>
  <DATA-GROUP>
   <DATA ref = "#personal"/>
   <DATA ref = "#medical">
    <CATEGORIES>
     <health/>
    </CATEGORIES>
   </DATA>
  </DATA-GROUP>
 </STATEMENT>
 ...
```

Figure 5

```
create restriction Statement1
on Patients
for public
to cells            Name, SSN, Address, Email, DOB,
                    XRay, Pharmacy, Family,
                    Appointment, Lifestyle
    purpose Emergency
    recipient ours
restricting access to select

create restriction Statement2.1
on Patients
for public
to cells            Name, SSN, Address, Email, DOB,
                    XRay, Pharmacy, Family,
                    Appointment, Lifestyle
      where
        exists (
          select 1
          from SysCat.Choices_Patients cp
          where cp.ID = Patients.ID
            and cp.C1 = 1 )
    for purpose develop
    for recipient ours
restricting access to select

create restriction Statement2.2
on Patients
for public
to cells            Name, SSN, Address, Email, DOB,
                    XRay, Pharmacy, Family,
                    Appointment, Lifestyle
      where
        exists (
          select 1
          from SysCat.Choices_Patients cp
          where cp.ID = Patients.ID
            and cp.C2 = 1 )
    for purpose develop
    for recipient same
restricting access to select
```

Figure 6

```
1  for each statement s in policy do begin
2     for each purpose p in s do begin
3        for each recipient r in s do begin

4           print "create restriction "
5           print generate-unique-restriction-name()
6           print " on table "
7           print mapP3PStatementToTable(s)
8           print " for public "
9           print " to cells "
10          print mapP3PDataTypeToColumns(s)

11          if (p.required != always) then
12             print "where exists (select 1 from "
13                   + mapP3PPurposeToChoiceTable(s)
14                   + " p where p.ID = "+ mapP3PStatementToTable(s) +".ID
15                   and "+ mapP3PPurposeToChoiceColumn(s,p) + "= 1))"

16          if (r.required != always) then
17             print "and exists (select 1 from "
18                   + mapP3PRecipientToChoiceTable(s)
19                   + " r where r.ID = "+ mapP3PStatementToTable(s) +".ID"
20                   + "and "+ mapP3PRecipientToChoiceColumn(s, r) + "= 1))"
21          print "for purpose" + p.name
22          print "for recipient" + r.name
23       end
24    end
25 end
26 print "restricting access to select"
```

Figure 7

EXTENDING RELATIONAL DATABASE SYSTEMS TO AUTOMATICALLY ENFORCE PRIVACY POLICIES

FIELD OF THE INVENTION

The present invention relates generally to the field of database systems. More specifically, the present invention is related to privacy preserving relational database management systems.

DISCUSSION OF PRIOR ART

The pervasive use of computing technology and the increased reliance on information systems have created a heightened awareness and concern about the storage and use of private information. This worldwide phenomenon has ushered in a plethora of privacy-related guidelines and legislations, e.g. the OECD Privacy Guidelines in Europe, the Canadian Privacy Act, the Australian Privacy Amendment Act, the Japanese Privacy Code, the Health Insurance Portability and Accountability Act (HIPAA), and Gramm-Leach-Bliley Consumer Privacy Rule. Compliance with these legislations has become an important corporate concern. The current methods employed to address the disclosure compliance problem involve training individuals to be cognizant of the various regulations and changing organizational processes and procedures. However, these approaches are only a partial solution and need to be augmented with technological support.

The users of relational databases require that a fine grained access control (FGAC) implementation meet the following desiderata:

- the implementation must solve the problem within the database itself without application changes or application awareness of the implementation.
- the implementation must ensure that all users of the data are covered, regardless of how the data is accessed.
- the implementation must minimize the complexity and maintenance of the FGAC policies.
- the implementation must provide the ability to control access to rows, columns, or cells as desired.

Traditional methods of database access control have relied upon the use of statically defined views, which are logical constructs defined over database tables that can alter or restrict the data seen by a user. Using predefined views as the method for FGAC works well only when the number of different restrictions is few or the granularity of the restrictions is such that it affects large, easily identified groups of users. When these conditions are not true, view definitions can become complex in an effort to accommodate all the restrictions in one view. This complexity can strain system limits and can make maintenance of views difficult.

If a large number of views are used, each one implementing restrictions for a specific set of users, one issue that arises is how to correctly route user requests to the view that is appropriate to them. Often, the solution chosen is to resolve the request in the application, not in the database. Moreover, if a user can bypass the view when accessing data, for example by having direct access to the underlying tables, then the restrictions are not enforced.

Given the shortcomings of the traditional methods of implementing FGAC, some database vendors have proposed solutions that do not rely on the use of views to control access to tabular data. For instance, Oracle™ Virtual Private Database solution as described in article titled, "Fine-grained access control" by Kyte and pages 240–253 of book

titled, "Oracle Privacy Security Auditing" by Nanda et al., allows users to define a security policy, which is a function written in PL/SQL that returns a string representing a predicate, and to attach the security policy to a table. When that table is accessed, the security policy is automatically enforced. In essence, row restrictions traditionally handled by views are allowed to be dynamically added to queries as described in article entitled, "Access control in a relational database management system by query modification", by Wong et al. The disadvantages of this approach are that Oracle™ requires user programming of a strictly defined "predicate producing" procedure in order to implement a security policy and it does not address column or cell restrictions. Sybase® Row Level Access Control as described in e-book entitled, "Sybase—Sybase Adaptive Server Enterprise 12.5, System Administration Guide", allows users to define access rules that apply restrictions to retrieved data. Sybase® Adaptive Server Enterprise 12.5 enables the database owner or table owner to restrict access to a table's rows by defining access rules and binding those rules to the table. Access to data can be further controlled by setting application contexts and creating login triggers. Access rules apply restrictions to retrieved data, enforced on select, update and delete operations. Adaptive Server enforces the access rules on all columns that are read by a query, even if the columns are not included in the select list. Using access rules is similar to using views, or using an adhoc query with where clauses. The query is compiled and optimized after the access rules are attached, so it does not cause performance degradation. Access rules provide a virtual view of the table data, the view depending on the specific access rules bound to the columns. Sybase® needs to create a separate access rule for each predicate, and'ing them, and then binding them to the appropriate columns. Microsoft® SQL Server primarily supports traditional view based access control, though it has a feature called row level permissions, but it seems to be usable only with table hierarchies. In IBM® DB2, the only support for FGAC is currently provided through the view mechanism.

The following references provide for creating views of datasets in database systems.

U.S. patent assigned to Microsoft Corporation, (U.S. Pat. No. 6,065,012), discloses rows and columns with data source control which will be asked for data in a particular cell. A dynamic summary view is generated by defined HTML page that links data binding HTML tables and other HTML controls to predetermined data within a storage of data. Accessing the subset of the program module is done at the cell level and may be done by executing a script to call defined methods of the objects within the program module or accessing a control module defined within the program module.

U.S. patent assigned to NCR Corporation, (U.S. Pat. No. 6,253,203), uses a large number of statically defined views to handle restrictions.

U.S. patent assigned to University of Minnesota, (U.S. Pat. No. 6,496,832), discloses a system for analyzing data organized into data sets and for transforming datasets into a visual representation. The visual representation appears to provide a dynamic view of cell structure and transformed data sets with the value of cells linked.

U.S. patent application publication assigned to International Business Machines Corporation, (2004/0215626 A1), discloses a method and system for improving performance

of database queries within an RDBMS system with metadata objects. The view of the data in support of one or more summary tables is automatically identified and adjusted.

Article entitled, "Query Evaluation Techniques for Large Databases", by Graefe, discloses enforcement of access control within a relational database environment.

Article entitled, "Hippocratic Databases" by Agrawal et al., discusses a vision of database systems that take responsibility for the privacy of data they manage, inspired by the Hippocratic Oath. The article also enunciates the key privacy principles that Hippocratic Databases should support.

Article entitled, "Limiting Disclosure in Hippocratic Databases" by LeFevre et al., discusses the incorporation of privacy policy enforcement into an existing application and database environment. Privacy policies (prescribed rule and conditions) are stored in the database where they can be used to enforce limited disclosure. Every query is associated with purpose and recipient pairs. SQL queries issued to the database are intercepted and augmented to reflect the privacy policy rules regarding the purpose and recipient issuing the query.

Whatever the precise merits, features, and advantages of the above cited references, none of them achieves or fulfills the purposes of the present invention.

SUMMARY OF THE INVENTION

The present invention provides for a method of providing fine grained access control within a database, the method comprising the steps of: receiving a user query; identifying and combining restrictions relevant to the user query, the restrictions specifying access to data in a table in the database at the level of at least one of or a combination of: individual rows, individual columns or individual cells, and the restrictions comprising a combination of access control and privacy policy restrictions; transforming the user query into an equivalent query which implements the restrictions; and accessing the data based on the equivalent query.

The present invention provides for a system providing fine grained access control (FGAC) within a database, wherein the system comprises a policy translator which accepts as input a least a privacy policy and privacy metadata catalogs; and a relational database which stores the privacy metadata catalogs and FGAC restrictions. The FGAC restrictions specify access to data in a table in the relational database at the level of at least one of or a combination of: individual rows, individual columns or individual cells, these restrictions comprising a combination of access control and privacy policy restrictions.

The present invention provides for an article of manufacture comprising a computer usable medium having computer readable program code embodied therein which provides fine grained access control within a database, the medium comprising: computer readable program code aiding in receiving a user query; computer readable program code identifying restrictions on access to data in a table in the database at the level of at least one or a combination of: individual rows, individual columns or individual cells, the restrictions comprising a combination of access control and privacy policy restrictions; computer readable program code transforming the user query into an equivalent query which implements the restrictions; and computer readable program code aiding in accessing the data based on the equivalent query.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates fine grained restriction syntax, as per the present invention.

FIG. **2** illustrates implementation architecture for constructs, as per the present invention.

FIG. **3** illustrates a method of providing fine grained access control within a database, as per the present invention.

FIG. **4** illustrates an algorithm for enforcing fine grained restrictions, as per the present invention.

FIG. **5** illustrates an example of a privacy policy for a healthcare provider, as per the present invention.

FIG. **6** illustrates the translation of a privacy policy into fine grained cell level restrictions, as per the present invention.

FIG. **7** illustrates an algorithm for translating a P3P privacy policy into fine grained cell level restrictions, as per the present invention.

BRIEF DESCRIPTION OF THE PREFERRED EMBODIMENTS

While this invention is illustrated and described in a preferred embodiment, the invention may be produced in many different configurations. There is depicted in the drawings, and will herein be described in detail, a preferred embodiment of the invention, with the understanding that the present disclosure is to be considered as an exemplification of the principles of the invention and the associated functional specifications for its construction and is not intended to limit the invention to the embodiment illustrated. Those skilled in the art will envision many other possible variations within the scope of the present invention.

Databases of the future must ensure the privacy of the data subjects whom they store information on. The security functionality offered by current commercial database products does not adequately address the key issues necessary to enforce privacy compliance: cell level policy enforcement. Compliance with current privacy legislation mandates that the user's consent be obtained for the use/disclosure of their personal information. Row or column level restrictions are not adequate for modeling scenarios where individuals may have opt-in/out choices with different aspects of their information. To achieve this goal of minimal disclosure while allowing useful tasks to be performed on relevant information, cell level enforcement is key. A similar case for cell level enforcement is made in article titled, "Limiting disclosure in Hippocratic Databases" by LeFevre et al.

The U.S. Department of Health and Human Services website provides a scenario requiring adherence to the HIPAA regulation. BlueCo is a healthcare provider that stores personal data on individuals who enroll in its plans. BlueCo has affiliations with a number of hospitals, research institutions, and marketing companies. Under HIPAA, any individually identifiable healthcare information held or transmitted by BlueCo is considered protected private information. For any use or disclosure of protected health information that is not for treatment, payment, or health care operation and that is not otherwise permitted (e.g. law enforcement), Blue Co must get the data subject's consent.

A simplified version of BlueCo's database is given in Table 1. ReasearchCo is an epidemiological research institute that periodically harvests BlueCo's data. Under HIPAA, all clients must give their consent for release of their home and office numbers.

TABLE 1

Table of BlueCo's clients

ID	Name	Home Phone	Work Phone	Salary
1	Alicia Campbell	408-418-5198	408-419-9111	10,000
2	Bob Bobbett	408-418-5198	408-419-9112	20,000
3	Carl Abrahams	408-333-6633	408-419-9113	30,000
4	Dan Charmer	408-432-8644	408-419-9114	40,000
5	Ellen Generous	408-555-1235	408-419-9115	50,000

Alicia Campbell opts out of having her home phone number, but does not mind if BlueCo discloses her office number. A researcher at Research Co issues the following query:

select name, homephone, officephone
from clients where salary<=30000

Given the choices that Alicia made, only her name and office phone number should be displayed as shown in Table 2.

TABLE 2

Cell Level Enforcement

Name	Home Phone	Office Phone
Alicia Campbell	—	408-419-9111
Bob Bobbett	408-418-5198	—
Carl Abrahams	408-333-6633	408-419-9113

Database systems employing row level controls restrict disclosure of all information in a particular row, when a restriction is only on particular columns in that row.

Thus, using conventional row level controls, the results for the query are those shown in Table 3. Both Alicia and Bob are no longer present in the result, even though they have agreed that one of their two phone numbers can be disclosed. This simple example illustrates the inadequacy of row level restrictions. Similar arguments can be made for column level restrictions. They are not flexible enough to allow disclosure of non-sensitive data and suppression of sensitive data on a subject by subject basis.

TABLE 3

Row Level Enforcement

Name	Home Phone	Office Phone
Carl Abrahams	408-333-6633	408-419-9113

The present invention presents constructs for imbuing relational database systems with fine grained access control and show how they can be used to enforce disclosure control enunciated in the vision for Hippocratic databases as described in article entitled, "Hippocratic Databases" by Agrawal et al. These constructs have been designed to fit well with the rest of the infrastructure of a relational database system. The present invention also provides for the implementation of proposed FGAC constructs. The present invention further describes how privacy policies written in a higher-level specification language such as P3P can be algorithmically translated into the proposed constructs.

Constructs defined according to the present invention, allow restrictions to be specified on the access to data in a table at the level of a row, a column, or a cell (i.e. individual column-row intersections). Privacy policies specified in high-level languages such as P3P can be translated into these constructs, or the policy could be specified directly using these constructs.

The proposed construct is complimentary to the current table level authorization mechanisms provided by commercial database systems using the 'grant' command as described in pages 122–128 of book entitled, "A complete Guide to DB2 Universal Database" by Don Chamberlin. While the 'grant' command controls whether a user can access a table at all, the constructs of the present invention define the subset of the data within a table that the user is allowed to access. Conceptually, a restriction defines a view of the table in which inaccessible data has been replaced by null values. As discussed in article entitled, "Limiting disclosure in Hippocratic Databases" by LeFevre et al, it is possible to use either "table semantics" or "query semantics". With query semantics, if all the values in a row are hidden by a restriction, then the row is omitted altogether from the view. With table semantics, the row would instead be retained unless a primary key column is restricted.

FIG. **1** gives the syntax of a fine grained restriction command, as per the present invention. It states that those in auth-name-1 except those in auth-name-2 are allowed only restricted access to table-x. As a short hand, the restriction can be defined for public (i.e., all users), and in that case the exception to all users can be provided in auth-name-2. The keywords group and user can be used to qualify the authorized names. FIG. **1**'s table-x can be any table expression.

A restriction, as per the present invention, presents a single command that comprises a combination of access control and privacy policy restrictions. A restriction can be specified at the level of a column, a row, or a cell. More than one restriction can be specified on a table for the same user. A restriction may also specify purposes and/or recipients for which the access is allowed. If no purpose or recipient is specified, then the restriction applies to all purposes and recipients respectively. If a purpose or recipient is specified, the user's access is limited to only the specified purpose-recipient combinations.

Akin to the database system variable user that can be referenced in queries and returns the id of the user issuing the query, the new system variables purpose and recipient return the list of purposes and recipients from the current query context. These values in turn determine the restrictions for the current query.

The command-restriction that appears as the last element of the syntax has the following form and states that access can be restricted to any combination of select, delete, insert, or update commands:

restricting access to (all|(select/delete/insert|update)+)

The Customer table with the following schema: Customer (id integer, name char(32), phone char(32)) is used below for illustration purposes.

Column Restriction:

A column restriction specifies a subset of the columns in table-x that auth-name-1 is allowed to access. The following restriction, named r1, ensures that only the id column of Customer is accessed by any database user:

create restriction r1
on Customer
for public
to columns id
restricting access to all

The restriction r2 below ensures that members of the account group and user Bob have only select access to columns name and phone.

```
create restriction r2
on Customer
for group acct, user Bob
to columns name, phone
restricting access to select
```

Row Restriction:

A row restriction gives the subset of rows in table-x that auth-name-1 is allowed to access. This subset is specified using a search-condition over table-x. The restriction r3 below ensures that every access to Customer is qualified by the predicate, name=user.

```
create restriction r3
on Customer
for public
to rows where name=user
restricting access to all
```

If user Bob issues select * from Customer, he would see id, name and phone for those rows where name equaled Bob.

Cell Restriction:

A cell restriction defines the row-column intersections that auth-name-1 is allowed to access. It is possible to specify multiple column-name lists, each possibly annotated with a search-condition. A search-condition is a correlated subquery with an implicit correlation variable t defined over the tuples of table-x. Access to the columns in column-name-list for each individual row identified by t is conditionally granted depending upon the result of the search condition. If no search-condition is given, then access is granted to all column values in column-name-list in table-x. If the search condition ignores the implicit correlation variable, then access is granted or denied to all columns values in column-name-list in table-x, depending upon the result of the search-condition.

The following is an example of a cell restriction used to enforce individual user's privacy preferences expressed as opt-in/out choices. Assume that for the purpose of marketing, Bob is allowed to see name, but his access to phone is allowed only if the user has opted-in to revealing her phone number.

```
create restriction r4
on Customer for user Bob,
to cells      name,
              (phone where exists (
              select 1
              from SysCat.Choices_Customer c
              where c.ID = Customer.ID and c.Cl = 1))
  for purpose marketing
  for recipient others
restricting access to select
```

The above restriction specifies cell restrictions for two column-name-lists: The first list contains the name column, and the second contains the phone column. The restriction allows Bob access to name, only if the variable purpose includes marketing, and recipient includes others. Otherwise, all values of the name column will be null for Bob.

The second list of columns has a search-condition associated with it since access to phone is dependent upon individual user choices. The search-condition comprises an existential subquery that uses the implicit correlation variable Customer. For each row in Customer, the subquery verifies, using the SysCat.Choices Customer table that stores individual opt-in/out choices, whether the user has opted-in for the disclosure of her phone number (represented by a column value of 1).

Combining Multiple Restrictions:

If multiple restrictions have been defined for a user u and a table T, then u's access to T is governed by the combination of these restrictions.

Assume initially that a user associates with a query a single purpose and a single recipient. Two design choices for combining multiple restrictions have been considered

Intersection—User u is allowed access to data defined by the intersection of all applicable restrictions. The details are shown in Table 4.

Union—User u is allowed access to data defined by the union of all applicable restrictions. The details are shown in Table 5.

TABLE 4

Combining Restrictions with Intersection

	row	column	cell
row	The search conditions of individual restrictions are and'ed together to define the intersection of rows accessible to a user.	The row restriction limits the rows accessible to the user. The column restriction further limits the columns within the rows accessible to the user.	The row restriction limits the rows accessible to the user. Within each row, the cell restriction further limits the access to the cells that qualify the cells' search condition.
column		The user's access is limited to those columns that appear in both of the column restrictions.	Column and cell restrictions intersect to limit access to only those columns that appear in both the restrictions. In addition, the cells restriction's search-condition further limits accessible cells within a column.
cell			The search-conditions are and'ed together and the user is allowed access to a cell if the composite condition is satisfied for the cell. The value of the composite condition for a cell that does not appear in both the restrictions is false.

TABLE 5

Combining Restrictions with Union

	row	column	cell
row	The search conditions of individual restrictions are or'ed together to define the union of rows accessible to a user.	The user is given access to all the cells for any row that satisfies the row restriction. Additionally, the user is allowed access to all the cells in any of the columns that satisfies the column restriction, irrespective of whether the corresponding rows satisfy the row restriction.	The user is given access to all the cells in any of the rows that satisfy the row restriction. Additionally, the user is allowed access to all other cells that satisfy the cell restriction's search-condition, irrespective of whether the corresponding rows satisfy the row restriction.
column		The user is allowed access to a column if it appears in either of the two column restrictions.	The user is given access to all the cells in any column appearing in the column restriction, regardless of whether the cell restriction is satisfied for these cells. For cells in a column for which the column restriction does not apply, access is given if the cell restriction is satisfied.
cell			The search conditions are or'ed together and the user is allowed access to a cell if the composite condition is satisfied for the cell.

If the commands specified in the command-restriction clauses of the restrictions being combined are different, they are respectively and'ed or or'ed depending upon the choice of intersection or union semantics.

Multiple restrictions can be combined in any order, both with intersection and union semantics. With the intersection semantics, the user's access to data decreases as additional restrictions are applied. Conversely, with union semantics, access to data increases as additional restrictions are applied.

Finally, if a query is annotated with multiple purpose recipient pairs, instead of a single pair, then restrictions governing access to any of the pairs become relevant for the query. These restrictions are then combined as above. Note that once a user's access to a table has been restricted, the user can only access the data allowed for the purposes and recipients specified in the restrictions.

A system for implementing the constructs of the present invention is shown in FIG. **2**. Cell level restrictions limited to select statement access are discussed in the remainder of the application; however, FGAC restrictions also apply to row and column level restrictions.

A policy translator **202** accepts a privacy policy **201** (written in, for example P3P) and metadata stored in privacy catalogs **206** in database **208** and generates restrictions that implement the policy. FGAC restrictions **204** are a combination of the privacy policy restrictions generated by policy translator **202** and access control restrictions that may be defined in the database. The FGAC restrictions relevant to individual queries annotated with purpose and recipient information **210** are identified and combined, and the user's query is transformed into an equivalent query over a dynamic view that implements the restriction. The schema of the privacy metadata catalogs shown in FIG. **2** used to drive the translation of P3P privacy policies into cell level restrictions are given below:

PR (purp-recip char(18),
p3ptype char(32),
choice tabname char(32),
choice colname char(32))

PT (p3ptype char (32), tabname char(32), colname char (32))

Table PR stores, for each purpose, recipient and p3p data type pair, the (table name-column name) pair that records individual user opt-in/out choice, should any choice be available for that combination. Table PT stores, for each P3P data type, the table names and column names which store values of these P3P types.

FIG. **3** is a flowchart illustrating an exemplary method as per the teaching of the present invention to provide fine grained access control within a relational database. A user query is received at step **300**. The user query is annotated with purpose and recipient information. FGAC restrictions which are a combination of privacy policy and access control restrictions are stored in the database. These FGAC restrictions may be specified at the level of individual rows, columns, cells, or a combination of these. In step **302**, the FGAC restrictions relevant to the user query are identified and combined. The user query is then transformed into an equivalent query over a dynamic view that implements the restriction, in step **304**. The data from the database is accessed based on the equivalent query, in step **306**.

FIG. **4** illustrates an exemplary algorithm, as per the teaching of the present invention, which enforces the fine grain restrictions. For ease of exposition, it is assumed that there is a single purpose-recipient pair associated with a query and there is at most a single restriction which is

relevant for the query. The enforcement algorithm combines the restrictions relevant to individual queries annotated with purpose and recipient information and transforms the user's query into an equivalent query over a dynamic view that implements the restriction.

In detail, Line 1 iterates over each table reference tin a query Q. Line 2 accesses metadata to determine if there is a restriction r governing the usage of t by user u who is submitting the query Q. If no such restriction exists, then t remains unmodified in Q. Otherwise, Lines 3 and 4 replace each reference to table tin query Q with a reference to a dynamic view v.

The generation of the dynamic view v is implemented in Lines 5 through 25. The view v is a select statement which conditionally projects each column cet. Line 7 searches for a column reference to cer. If no such reference exists with the purpose/recipient of query Q, then the user u is not allowed access to c and Line 8 thus projects a null value for all values of c. Otherwise, Line 10 searches for a where clause associated with cer. If no such clause exists, then u is granted unconditional access to c. Otherwise, Line 15 outputs the condition of the where clause into a SQL case statement which verifies the condition before outputting the value of c (on Line 18). If the condition is false, access to the column value is denied and Line 19 outputs a null value for c.

The following illustrates the basic syntax of the P3P policy specification language:

```
<POLICIES> . . .
    <POLICY name = "Policy_Name1"> . . .
        <STATEMENT>
        . . .
            <PURPOSE>
                stated-purpose
                    [ required = ("always"|"opt-in"|"opt-out") ]
            </PURPOSE>
            <RECIPIENT>
                stated-recip
                    [ required = ("always"|"opt-in"|"opt-out") ]
            </RECIPIENT>
            <RETENTION> retention_val </RETENTION>
            <DATA GROUP>
                <DATA ref = data-ref-val>
                . . .
            </DATA GROUP>
        </STATEMENT>
    </POLICY>
    <POLICY>
    . . .
    </POLICY>
    . . .
</POLICIES>
```

The process of transforming a policy like the one above into fine grained restrictions involves: (1) parsing the policy to extract the list of statements, (2) mapping data abstractions into their implementation specific equivalents, e.g. in the above specification this would mean mapping data-ref-val to its corresponding table name(s) and column name(s), (3) structuring the choice tables which record individual user opt-in/out choices (in some cases, this may not be necessary since there may be no such choices), and (4) generating the restriction statements. Assuming that data-ref-val maps to columns A and B of table T, the above abstract specification would lead to the following restriction being constructed:

```
create restriction Policy Name 1
on T
for public
to cells A,B
[where opt-in-out-conditions]
for purpose stated-purpose
for recipient stated-recip
restricting access to select
```

FIG. **5** is a detailed example of a privacy policy, for a fictional Healthcare provider. The metadata contains the information needed to associate "#personal" (personal information) and "#medical" (medical information) with database tables which store this information. Personal information maps to the name, SSN, address, email and DOB fields of the Patients table, while medical information maps to the xray, pharmacy, family, appointment and lifestyle fields of the Patients table. Physician, Healthcare and Drug_Research are assumed to be user roles and thus do not require refining. Thus, the P3P healthcare policy given in FIG. **5** is translated into the restrictions given in FIG. **6**. For simplicity, the restrictions in FIG. **6** assume that all columns in a P3P policy are contained in a single table.

The creation and population of the Choices_Patients table should be coordinated to synchronize with the creation and update of the patients table. The policy translator modifies the structure of the choices_patients table to ensure that the correct number of choice fields are present for recording opt-in/opt-out decisions for a particular table. In the above example, C1 represents the choice to allow Drug_Research to see personal data if the drug research is being conducted by the healthcare company itself. Choice C2 is the option to allow usage of personal data for drug research by other healthcare companies having the same privacy policy as this company. The example illustrates the basic steps involved in the translation process. FIG. **7** gives the pseudo-code showing the steps involved in transforming P3 policy into the present invention's language constructs.

A unique restriction name, needed for the command is generated on Line 2. Line 3 uses the mapP3PPolicyToTable function to uncover the table name which stores the information described by the data types in the P3P statement. This metadata is populated by the database administrator. On Line 4, the set of users who are authorized to access data specified by the policy are obtained using the mapP3PPolicyToAuthorizedUsers command which uses database metadata to derive the set of authorized users. The database administrator is responsible for populating this information in the database metadata tables. Line 10 uses the mapP3PDataTypeToColumns function to retrieve the column names that store information described by the P3P data types in the statement. Again, this information has been prepared and supplied by the database administrator and stored in metadata tables.

The function mapP3PPurposeToChoiceTable accepts a statement id and returns the table storing individual user choices for this statement. The function mapP3PPurposeToChoiceColumn accepts a statement-purpose pair and returns the column in the choice table which records the corresponding users' choices. Both these functions are driven from metadata.

Although the present invention, as described, provides restrictions specified for tables and at least one or a combination of rows, columns or cells in a relational database; it should be noted that restrictions can also be specified for collection of objects and attributes of these objects in an object database, or collection of documents and attributes of elements in these documents in an XML database. Hence, how such restrictions are specified should not be used to limit the scope of this invention.

Additionally, the present invention provides for an article of manufacture comprising computer readable program code contained within implementing one or more modules to provide fine grained access control in a relational database. Furthermore, the present invention includes a computer program code-based product, which is a storage medium having program code stored therein which can be used to instruct a computer to perform any of the methods associated with the present invention. The computer storage medium includes any of, but is not limited to, the following: CD-ROM, DVD, magnetic tape, optical disc, hard drive, floppy disk, ferroelectric memory, flash memory, ferromagnetic memory, optical storage, charge coupled devices, magnetic or optical cards, smart cards, EEPROM, EPROM, RAM, ROM, DRAM, SRAM, SDRAM, or any other appropriate static or dynamic memory or data storage devices.

Implemented in computer program code based products are software modules for:

(a) aiding in receiving a user query;

(b) identifying and combining restrictions relevant to the user query, the restrictions specifying access to data in a table in the database at the level of at least one or a combination of: individual rows, individual columns or individual cells, and the restrictions comprising a combination of access control and privacy policy restrictions;

(c) transforming the user query into an equivalent query which implements the restrictions; and

(d) aiding in accessing the data based on the equivalent query.

CONCLUSION

A system and method has been shown in the above embodiments for the effective implementation of extending relational database systems to automatically enforce privacy policies. While various preferred embodiments have been shown and described, it will be understood that there is no intent to limit the invention by such disclosure, but rather, it is intended to cover all modifications falling within the spirit and scope of the invention, as defined in the appended claims. For example, the present invention should not be limited by software/program, computing environment, or specific computing hardware. Moreover, the present invention should not be limited to how the restrictions are specified. All programming and data related thereto are stored in computer memory, static or dynamic, and may be retrieved by the user in any of: conventional computer storage, display (i.e., CRT) and/or hardcopy (i.e., printed) formats.

What is claimed is:

1. A method of providing fine grained access control within a database, said method comprising:

receiving a user query;

identifying and combining restrictions relevant to said user query, said restrictions specifying access to data in a table in said database at the level of at least one of or a combination of: individual rows, individual columns or individual cells, and said restrictions comprising a combination of access control and privacy policy restrictions, said privacy policy restrictions being generated by transforming a privacy policy by the following steps:

parsing said privacy policy to extract a list of statements,

mapping data abstractions in said privacy policy into their implementation specific equivalents,

structuring choice tables which record individual user opt-in/out choices, and

generating restriction statements;

transforming said user query into an equivalent query which implements said restrictions; and

accessing said data based on said equivalent query.

2. A method of providing fine grained access control within a database, according to claim **1**, wherein said restrictions are generated by transforming said privacy policy and using privacy metadata catalogs.

3. A method of providing fine grained access control in a database, according to claim **1**, wherein said restrictions are combined by union or intersection.

4. A method of providing fine grained access control in a database, according to claim **2**, wherein said privacy policy is written in high-level policy language.

5. A method of providing fine grained access control in a database, according to claim **4**, wherein said high-level policy language is P3P.

6. A method of providing fine grained access control in a database, according to claim **2**, wherein said privacy metadata catalogs store individual opt-in/opt-out choices.

7. A method of providing fine grained access control in a database, according to claim **1**, wherein said restrictions specify purposes and/or recipients for which access is allowed.

8. A system providing fine grained access control (FGAC) within a database, said system comprising:

a database to store privacy metadata catalogs and FGAC restrictions, said FGAC restrictions specifying access to data in a table in said database at the level of at least one of or a combination of: individual rows, individual columns or individual cells and said FGAC restrictions comprising a combination of access control and privacy policy restrictions, said data of said database being accessed based on a transformed equivalent query which implements said FGAC restrictions; and

a policy translator to accept as input a least a privacy policy and said privacy metadata catalogs, said policy translator transforming said privacy policy into said privacy policy restrictions by: parsing said privacy policy to extract a list of statements, mapping data abstractions in said privacy policy into their implementation specific equivalents, structuring choice tables which record individual user opt-in/out choices, and generating restriction statements.

9. A system providing fine grained access control (FGAC) within a database, according to claim **8**, wherein said privacy policy is written in high-level policy language.

10. A system providing fine grained access control (FGAC) within a database, according to claim **9**, wherein said high-level policy language is P3P.

11. A system providing fine grained access control (FGAC) within a database, according to claim **8**, wherein said privacy metadata catalogs store individual opt-in/opt-out choices.

12. A system providing fine grained access control (FGAC) within a database, according to claim **8**, wherein said FGAC restrictions specify purposes and/or recipients for which access is allowed.

13. An article of manufacture comprising a computer usable medium having computer readable program code embodied therein which provides fine grained access control within a database, said medium comprising:

computer readable program code aiding in receiving a user query;

computer readable program code identifying restrictions on access to data in a table in said database at the level of at least one or a combination of: individual rows, individual columns or individual cells, said restrictions comprising a combination of access control and privacy policy restrictions;

computer readable program code transforming a privacy policy into said privacy policy restrictions by: parsing said privacy policy to extract a list of statements, mapping data abstractions in said privacy policy into their implementation specific equivalents, structuring choice tables which record individual user opt-in/out choices, and generating restriction statements;

computer readable program code transforming said user query into an equivalent query which implements said restrictions; and

computer readable program code aiding in accessing said data based on said equivalent query.

* * * * *

Appendix B: Data Encryption Portfolio

Table B.1 – Research Papers

ID	Publication	Key Contributions
1	**Holistic Database Encryption** *International Conference on Security and Cryptography (SECRYPT)*	- Design of a holistic database encryption solution which allows organizations to meet their security and compliance requirements without having to make compromises either on the security side or on the database side. - Enable organizations to adhere to zero-trust security. - Implementation of the solution in IBM DB2 for Linux, Unix and Windows.
2	**Towards Zero-Trust Database Security – Part 1** *IEEE Future Directions Newsletter: Technology Policy & Ethics*	- Introduces a database threat model and raises awareness of the direct and indirect means through which the same data in a database can be accessed.
3	**Towards Zero-Trust Database Security – Part 2** *IEEE Future Directions Newsletter: Technology Policy & Ethics*	- Outlines solutions (including encryption) to address the direct and indirect access challenges and to enable zero-trust database security.

Holistic Database Encryption

Walid Rjaibi
IBM Canada Lab, 8200 Warden Avenue, Markham, Ontario, Canada
wrjaibi@ca.ibm.com

Keywords: Databases, Encryption, Key Management, Security, Compliance.

Abstract: Encryption is a key technical control for safeguarding sensitive data against internal and external threats. It is also a requirement for complying with several industry standards and government regulations. While Transport Layer Security (TLS) is widely accepted as the standard solution for encrypting data in transit, no single solution has achieved similar status for encrypting data at rest. This is particularly true for database encryption where current approaches are forcing organizations to compromise either on the security side or on the database side. In this paper, we discuss the design and implementation of a holistic database encryption approach which allows organizations to meet their security and compliance requirements without having to sacrifice any critical database or security properties.

1 INTRODUCTION

Internal threats, external threats, government regulations, and industry standards require organizations to implement security controls to ensure information is adequately protected. Failure to do so can have a negative impact on an organization such as loss of customer data, damage to brand reputation, and even financial penalties. Encryption is a key technical control for protecting information. It is also an explicitly stated requirement for compliance with many regulations and standards such as the General Data Protection Regulation (Voigt, 2017) and the Payment Card Industry Data Security Standard (Chuvakin, 2009).

While TLS is widely accepted as the standard solution for encrypting data in transit, no single solution has achieved similar status for encrypting data at rest. This is particularly true for database encryption where current approaches are forcing organizations to compromise either on the security side or on the database side. Indeed, database encryption poses some very unique challenges as not only the solution needs to be sound from a security perspective, but it also needs to coexist in harmony with critical database properties such as performance, integrity, availability, and compression.

The rest of this paper is organized as follows. Section 2 discusses the related work around database encryption. In section 3, we state our contributions. Section 4 defines the threats our database encryption solution defends against. In section 5, we describe our solution design in full details. Lastly, section 6 summarizes our approach and outlines our future work.

2 RELATED WORK

Current database encryption solutions can be divided into four main categories: Column encryption (Benfield, 2001), tablespace encryption (Freeman, 2008), file system encryption (Anto, 2018), and self-encrypting disks (Dufrasne, 2016). Unfortunately, each of these solutions forces the organization to make a compromise either on the database side or on the security side.

Column encryption negatively affects database performance as queries with range predicates cannot benefit from index-based access plans to limit the data to read from the table. Instead, the database system is forced to read the entire table to evaluate the query. Tablespace encryption may leave certain data vulnerable to attacks when, for example, an administrator inadvertently takes an action that moves data from an encrypted tablespace to an unencrypted one. An example of such action would

be the creation of a materialized query table (MQT) to speed up the execution of data warehousing queries. File system encryption and self-encrypted disks provide no protection against privileged users on the operating system. As long as the file permissions allow access, such users can easily view the content of the database by browsing the underlying files on the operating system.

3 CONTRIBUTIONS

The crux of our contribution is the design of a holistic database encryption approach which allows organizations to meet their security and compliance requirements without having to make compromises either on the security side or on the database side. Our solution improves over the state of the art discussed above as follows:

- Pervasiveness: All data is encrypted whether it is user tablespace data, system tablespace data, temporary tablespace data, transaction logs data, or database backups data.
- Security: The database content is not vulnerable to attacks by malicious administrators who may choose to bypass the database and access the database indirectly through the file system interfaces.
- Performance: The database system is not forced to dismiss index-based access plans to answer queries with range predicates.
- Breadth: The solution is built into the database engine itself which means that it is available on all platforms where the database system itself runs. Also, it does not force the database system to dismiss the opportunity to bypass the file system and write data directly to raw devices in order to boost performance.
- Quantum-safety: The implementation does not make use of asymmetric encryption to wrap data encryption keys. Data encryption keys are wrapped with symmetric encryption (Chandra, 2014). Therefore, the implementation is safe against future attacks by quantum computers implementing Shor's algorithm which is known to break asymmetric encryption that is based on integer factorization such as RSA or on discrete logarithms such as Diffie-Hellman (Shor, 1997). Additionally, the default encryption key size is 256 bits. This also makes the implementation safe against future attacks by quantum computers implementing Grover's algorithm which is known to offer a quadratic improvement in brute-force attacks on symmetric encryption schemes like AES (Grover, 1996).

We have also implemented the solution in a commercial database system (IBM DB2 for Linux, Unix, and Windows).

4 THREAT MODEL

We focus on protecting data at rest. For protecting data in transit between a database server and a client application against eavesdroppers, we assume TLS has been configured to provide this protection. TLS is the standard for protecting data in transit and is implemented by all major database systems.

The content of a database deployed on a given database server can be accessed in two different ways: Directly and indirectly. Direct access is when users interact with the database using the usual database interfaces such as querying the database tables using SQL. In this context, we assume that the database authentication and authorization mechanisms have been configured to ensure that data is accessible only to the appropriate users. Authentication ensures that users are who they claim they are while authorization ensures that authenticated users have access only to those objects or elements within objects for which they have been granted permissions (Rjaibi, 2004).

Indirect access is when a user chooses to bypass the database system altogether and uses operating system commands to browse the content of the database. For example, on Linux, the following command would display the content of the physical file associated with a given tablespace:

```
strings
'/u01/database/payroll_tbspace'
```

This command will be executed by the operating system bypassing all the database authentication and authorization controls.

Our solution addresses this threat by encrypting the database and ensuring that such encryption is under the control of the database system itself. This means that if a user chooses to bypass the database system as shown above, the operating system command will return cipher text which will be of no value to the attacker.

An attacker may also choose to access the database content from decommissioned hard drives or by physically stealing such hard drives. Our solution addresses this concern as well because the attacker will only find cipher text on those drives. Figure 1 gives a high level overview of our database threat model.

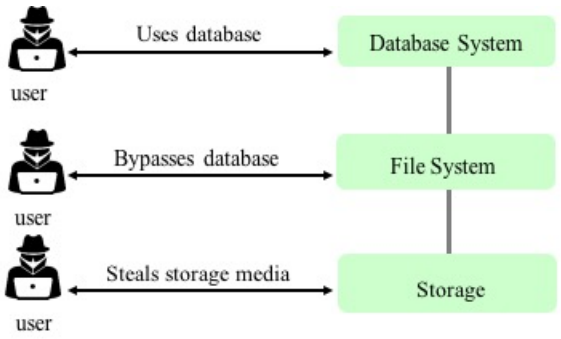


Figure 1: Database threat model.

5 DATABASE ENCRYPTION DESIGN

5.1 Encryption Key Management

Encryption key management is a critical aspect of an encryption solution. Our solution uses two types of encryption keys: A Data Encryption Key (DEK) and a Master Key (MK).

The DEK is the encryption key used to encrypt the actual data in the database. It is automatically generated by the database system at database creation time. The DEK is encrypted with the MK and stored within the database configuration structures together with the following attributes:

- The encryption key size: This is the length of the encryption key in bits (e.g., 256 bits).
- The encryption algorithm: This is the symmetric encryption algorithm used to encrypt the data with the DEK (e.g., AES).
- The master key label: This is the unique identifier of the master key within the external management system. For example, if the external management system is a Hardware Security Module (HSM), then the database system will call out to the HSM and ask it to either encrypt or decrypt the DEK as required. A call to decrypt the DEK is done once when the database system starts up. A call to encrypt the DEK is also done once when the database is created.
- The master key integrity value: To guard against the (rare) event where the MK acquired at some future point in the life of the database is not the one that was actually used to encrypt the DEK, we calculate an integrity value for the MK. We do this by applying a Hash Message Authentication Code (HMAC) function to the MK and store the result. Before making use of the DEK, we first compute an HMAC based on the MK acquired. If the computed HMAC and the stored HMAC match this implies that the master key acquired is indeed the one that was used to encrypt the DEK. Although rare, this is important to avoid corrupting data through decryption with the wrong key.

The MK is the encryption key used to encrypt the DEK. Only a unique identifier of the MK is stored within the database configuration structures. The MK itself is stored in an external key management system such as an HSM.

The reasons for choosing these two types of keys are security, performance, and availability. By storing the MK physically away from the database system, we are assured that compromise of the database system infrastructure does not give the attacker access to both the encrypted data and the encryption keys. Additionally, the concept of MK allows database administrators to rotate encryption keys without impacting the database performance or worse requiring the database to be taken offline to complete the operation. In fact, rotating the MK only requires decrypting the DEK with the old MK and re-encrypting it again with the new MK. In contrast, rotating the DEK requires reading the whole database, decrypting the data with old DEK, re-encrypting it with the new DEK, and writing it back to disk. Thus, the two types of keys we chose in our solution design (DEK and MK) allow administrators

to meet their regulatory compliance needs around rotating encryption keys without necessarily having to incur a performance penalty or take a downtime.

5.2 Data Encryption

Implementing security in database systems is always a delicate balance between meeting the security requirements, and ensuring that security coexists in harmony with other critical database features such as performance, compression, and availability. For database encryption, this means that the placement of the encryption run-time processing is key to designing an effective solution.

5.2.1 Encryption Run-time Placement

Our design places the encryption run-time processing just above the database I/O layer in the database kernel stack. The reasons for this choice are the following:

- Pervasiveness: This ensures that all data is encrypted whether it is user tablespace data, system tablespace data, temporary tablespace data, or transaction logs data.
- Transparency: This ensures that encryption has no impact on database schemas and user applications. In fact, encryption can be thought of as invisible to them.
- Performance: This ensures that data stored in the database buffer cache remains in clear text. Consequently, encryption imposes no restrictions on the database system when it comes to selecting the most efficient plan to execute a query (e.g., queries with range predicates).
- Compression: Database systems implement compression techniques to reduce the size of the data stored on disk. Typically, these techniques look for repeating patterns in order to avoid storing all copies of such patterns. Encryption, by definition, removes all patterns. This means that the order in which compression and encryption are performed is important. For example, if encryption is performed first, then the compression rate will be zero as encryption will leave no patterns. Thus, placing our encryption run-time processing just above the database I/O layer ensures that encryption and compression can coexist in harmony.

5.2.2 Encryption Run-time Processing

The encryption run-time processing consists of two functions: Encryption and decryption. Encryption takes place when the database system is writing data out to the storage system. Decryption happens when the database system is reading data in from the storage system.

While the solution can easily support any symmetric block cipher for encryption/decryption, we have chosen to implement support for only AES and 3DES as they are the most commonly used block ciphers. AES is actually the standard symmetric block cipher. Block ciphers support many modes of operations. Electronic Code Book (ECB) is the easiest mode to implement but is also the weakest from a security perspective. This is because in ECB mode the same clear text input will always result in the same cipher text. This may be fine for encrypting small pieces of data such as a password, but not for database encryption as this will introduce patterns and may compromise the encryption solution. Instead, we have chosen to use the Cipher Block Chaining (CBC) mode as it does not introduce patterns. This means we need to provide an initialization vector when calling the block cipher in CBC mode for encryption, as well as maintain that initialization vector in our meta-data so that it is available for decryption purposes. Note that the initialization vector is not meant to be a secret. It only needs to be random.

When writing data to the file system, the database system writes them in chunks to minimize the I/O overhead. A chunk is a collection of data pages where each page is 4KB in size. A page is set of rows, and a database table is a collection of pages. This poses an interesting question as to the level of granularity to adopt for encryption. We have chosen the data page to be that level granularity. A row level granularity would have had a higher impact on performance as encryption calls would have to be made for each row separately. A chunk level granularity would have created a dependency between the pages in that chunk due to the chaining inherent to the CBC mode. For example, to decrypt page 5, one must first decrypt pages 1, 2, 3, and 4. This would have had a negative impact on query performance as it diminishes the value of index-based access.

It is also worth noting that the data page level granularity has allowed us to avoid having to

needlessly increase the database size due to encryption. In fact, encryption block ciphers such as AES and 3DES encrypt data one block at a time. For example, the block size for AES is 16 Bytes. This means that when the clear text to encrypt is not an exact multiple of the block size, padding is required and this obviously increases the cipher text compared to the original clear text. Fortunately, the choice of a data page for the encryption granularity avoids this problem as data pages are always an exact multiple of the encryption block size.

5.2.3 Transaction logs

Transaction logs are files where the database system logs transactions such as insert, delete, and update operations. They are a critical component for ensuring the integrity of the database as well as for allowing recoverability of the database following a database crash. The structure of a transaction log file consists of two pieces: A header which contains meta-data about the file, and a payload which contains the actual database transaction details.

In section 5.2.2 above, we have seen how the placement of the encryption run-time ensures that all data written to disk, including transaction logs, is automatically encrypted. However, transaction logs pose one additional challenge. In a database recovery scenario, we must be able to decrypt the transaction logs even when the database system is down. This means that we cannot rely on the DEK related information (section 5.1 above) to decrypt the transaction logs as the database system may be offline. To address this challenge, the transaction logs structure has been extended so that these logs are self-contained when decryption is required. More specifically, the header piece of the transaction logs structure has been extended so that it contains its own copy of the DEK related information. This also opens the door for an opportunity to further boost security by generating a separate DEK for the transaction logs that is distinct from the DEK for the database.

5.2.4 Database backups

A database backup is a copy of the database content at a given point in time. Database systems provide a command and/or API to allow users to take those backups. In the case of a database crash, the database can be recovered to the state it was at when the last backup was taken. Additionally, when healthy transaction logs from the damaged database are available, it is possible to recover the database to a further point in time by reapplying the database transactions from the transaction logs. Like transaction logs, a database backup consists of two pieces: A header which contains meta-data about the backup, and a payload which contains the actual copy of the database.

Database backups pose the same challenge as transaction logs in the sense that they too need to be self-contained when decryption is required. Consequently, this challenge is addressed in the same way by extending the database backup header piece so that it contains its own copy of the DEK related information. Like transaction logs, database backups have their own unique DEK.

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented a holistic approach to database encryption which allows organizations to meet their security and compliance needs without having to make compromises either on the security side or on the database side. Figure 2 gives a high level overview of the architecture, which we implemented in IBM DB2 for Linux, Unix, and Windows.

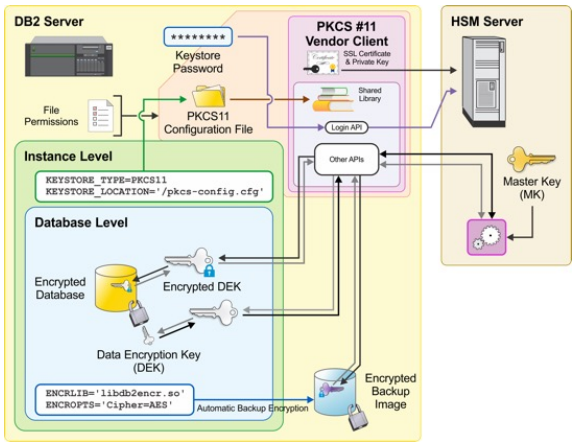


Figure 2: Database encryption architecture.

In our future work, we intend to enhance our holistic database encryption solution to better address two challenges. The first challenge is encrypting existing databases. Unlike newly created databases, an existing database already has data and turning encryption on for that database means not

only encrypting new incoming data, but also encrypting that existing data. The current solution requires the organization to turn on the encryption for the existing database during a scheduled database maintenance window. This is because the current approach for encrypting an existing database works by having the database administrator take a backup of the existing database and then restoring it using the RESTORE DATABASE command. While processing the restore, the database system encrypts the data as that is analogous to new incoming data. We would like to allow database administrators to turn on encryption for their existing databases without having to wait for a scheduled maintenance window. To do so, we plan to investigate creating a background process which encrypts the database incrementally while the database system continues to serve applications. The main challenge would be finding out how to perform this incremental encryption without compromising the data integrity.

The second challenge is rotating the DEK online. Currently, our solution allows rotating only the MK online. While rotating the MK is usually sufficient, there may be situations where rotating the DEK itself is required. Currently, the only way to do this is during a scheduled maintenance window following the same database backup and restore discussed above. We believe that the solution for encrypting existing databases without having to wait for scheduled maintenance window would also allow rotating the DEK online as that is fundamentally the same problem. That is, in both cases, the database content needs to be read, re-encrypted with a new DEK, and written back to disk.

ACKNOWLEDGEMENTS

The author would like to thank Saifedine Rjaibi and Devan Shah for their valuable comments.

REFERENCES

Rjaibi, W., Bird, P., 2004. A Multi-Purpose Implementation of Mandatory Access Control in Relational Database Management Systems. In *VLDB'04, 30th International Conference on Very Large Data Bases*. Morgan Kaufmann.

Chandra, S., Paira, S., Alam, S., Sanyal, G., 2014. A Comparative Survey of Symmetric and Asymmetric Key Cryptography. In *ICECCE'14, International Conference on Electronics, Communication and Computational Engineering*. IEEE.

Grover, L., 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *STOC'96, 28th Annual ACM Symposium on Theory of computing*. ACM.

Shor, P., 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM Journal on Computing, Volume 26 Issue 5.

Dufrasne, B., Brunson, S., Reinhart, A., Tondini, R., Wolf, R., 2016. *IBM DS8880 Data-at-rest Encryption*, IBM Redbooks. New York, 7th edition.

Benfield, B., Swagerman, R., 2001. Encrypting Data Values in DB2 Universal Database. IBM DeveloperWorks.

Anto, J., 2008. Understanding EFS. IBM DeveloperWorks.

Freeman, R., 2008. *Oracle Database 11g New Features*, McGraw-Hill.

Voigt, P., Von Dem Bussche, A., 2017. *The EU General Data Protection Regulation (GDPR)*, Springer International.

Chuvakin, A., Williams, B., 2009. *PCI Compliance: Understand and Implement Effective PCI Data Security Standard Compliance*, Elsevier.

Towards Zero-Trust Database Security – Part 1

Walid Rjaibi, Mohammad Hammoudeh

Abstract—The rise of external threats, internal threats and data breaches is driving enterprises to implement zero-trust security to better protect their IT assets and reduce risk. While zero-trust security for networks and identity management systems have received a great deal of focus, very little attention has been devoted to zero-trust security for database systems. This is a major issue as database systems are the custodian of enterprises most critical data and are often the primary target of both external and internal attacks. After all, databases contain valuable data such attackers want to steal. In Part One of this series, we explore both the direct and indirect means through which the same data in a database system can be accessed and the challenges they pose to adhering to the basic tenets of zero-trust security. In Part Two, we outline a set of solutions that are most suitable to address these challenges and enable enterprises to implement zero-trust database security without negatively impacting core database tenets such as query performance.

Index Terms—Databases, Security, Zero-Trust.

1 Introduction

THE 2018 Cost of a Data Breach Study, conducted by the Ponemon Institute and sponsored by IBM, found that the global average cost of a data breach was $3.86 million [1]. This was an increase of 6.4% compared to 2017 according to the same study. The study also found that the average size of a data breach (in terms of number of records lost or stolen) grew 2.2% from 2017. Meanwhile, Gartner estimates that worldwide spending on cybersecurity in 2018 was around $114 billion, an increase of 12.4% compared to 2017 [2]. Recognizing that current approaches aren't sufficiently adequate, several organizations are now turning into zero-trust security to better protect their assets and reduce the risk of incurring a data breach. So, what exactly is zero-trust security?

Zero-trust security was coined by Forrester's John Kindervag in 2010 [3], [4]. In its essence, zero-trust security removes the notion of trust from the enterprise network (e.g., no more trusted users, devices, or applications). It assumes that untrusted entities exist both outside and inside the enterprise network. The basic tenets of zero-trust security can be summarized as follows:

1. Tenet 1: Ensure all resources are accessed in a secure manner regardless of location.
2. Tenet 2: Grant access to resources based on "need-to-know" and strictly enforce access control.
3. Tenet 3: Monitor and audit all user activities.

While extensive coverage of zero-trust security implementations for networks [3] and identity management systems [5] exists, very little coverage exists for database systems. We contend that zero-trust security implementations for database systems is equally important for three main reasons. First, database systems are the custodians of the enterprise most valuable data. This is the very data attackers of all sorts are seeking. Secondly, the same data entrusted with the database system can be accessed in a variety of distinct and independent ways, thus broadening the database attack surface. Lastly, the database privileges model is inherently a double-edged sword, creating opportunities for privileges to be abused intentionally or unintentionally.

2 Database Threat Model

We assume that organizations are implementing user authentication, auditing and Transport Layer Security (TLS) which are standard features on all major database systems today. We also assume that organizations are implementing adequate operational policies such as operating system and database software vulnerability patching. In this paper, we focus on direct and indirect means for accessing data in a database and the challenges they pose to adhering to the three zero-trust security tenets outlined in section 1.

The same data in a database can be accessed in two different ways: Indirectly or directly. Indirect access occurs when a user bypasses the database system altogether. This is most dangerous because it completely bypasses all database access control and auditing. We distinguish between two use cases:

1. File system access: This takes place when a user chooses to access the data directly on the file system using operating system com-

- *Walid Rjaibi is with the Departmentof Computing and Mathematics, Manchester Metropolitan University, and the IBM Canada Lab, 8200 Warden Avenue, Markham ON L6E 1E9. E-mail: wrjaibi@ca.ibm.com.*
- *Mohammad Hammoudeh is with the Department of Computing and Mathematics, Manchester Metropolitan University, Manchester M15 6BH. E-mail: M.Hammoudeh@mmu.ac.uk.*

mands.

2. Storage media access: This takes place when a user recovers the data from the actual storage media such as a stolen or lost hard drive or tape.

Failure to address these two use cases makes it impossible to adhere to the first two tenets of zero-trust security outlined in section 1.

Direct access takes place using standard database interfaces such as Structured Query Language (SQL). We distinguish between two use cases:

1. Interactive database access: This is typically done by database administrators using an interactive interface offered by the database system. This is usually used to perform administrative tasks.
2. Application database access: This is the most common use case where end users interact with an application which in turn interacts with the database system to execute requests on behalf of those end users.

The issue with interactive database access is *privilege abuse* where, for example, a database administrator chooses to abuse their privileges to access sensitive data. The application database access poses two issues. The first one is *application bypass* where, for example, the application administrator chooses to abuse the application database credentials to access sensitive data or make changes that are not permitted by the application's business logic. The second issue is the *loss of user identity* which diminishes the value of auditing to hold users accountable for their actions. This stems from the fact that the application uses a generic user ID to access the database on behalf of all users as opposed to the actual user identity.

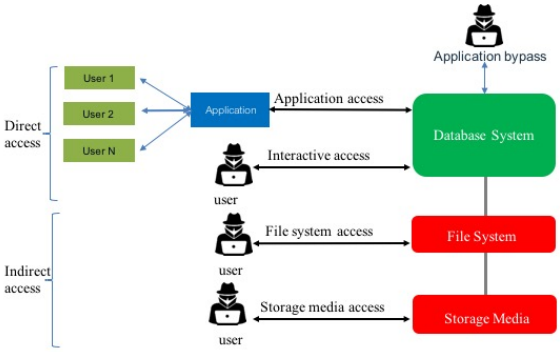


Fig. 1. Database threat model.

Failure to address privilege abuse and application bypass makes it impossible to adhere to the first two tenets of zero-trust security outlined in section 1. Similarly, failure to address the loss of user identity makes it impossible to adhere to the third tenet of zero-trust security (aslo outlined in section 1). Fig. 1 summarizes our database threat model.

3 Conclusion

Database systems contain enterprises most valuable data and are often the primary target of both internal and external attacks. Implementing zero-trust database security starts with first understanding the database threat model. Table 1 summarizes these threats and how they relate to the basic tenets of zero-trust security. In Part Two of this series we outline solutions and best practices for addressing these threats and implement zero-trust database security.

TABLE 1
ZERO-TRUST DATABASE SECURITY CHALLENGES

Threat	Threat type	Fundamental zero-trust security tenet
File system access	Indirect	Tenets 1 and 2
Storage media access	Indirect	Tenets 1 and 2
Privilege abuse	Direct	Tenets 1 and 2
Application bypass	Direct	Tenets 1 and 2
Loss of end user identity in multitiered environments	Direct	Tenet 3

References

[1] The Ponemon Institute, https://www.ibm.com/security/data-breach, 2019.

[2] Gartner, https://www.gartner.com/en/newsroom/press-releases/2018-08-15-gartner-forecasts-worldwide-information-security-spending-to-exceed-124-billion-in-2019, 2019.

[3] E. Gilman, D. Barth, *Zero Trust Networks: Building Secure Systems in Untrusted Networks*. O'Reilly Media, 2017.

[4] S. Walker-Roberts, M. Hammoudeh, A. Dehghantanha, "A Systematic Review of the Availability and Efficacy of Countermeasures to Internal Threats in Healthcare Critical Infrastructure", *IEEE Access*, 6, pp.25167-25177, 2018.

[5] Centrify, https://www.centrify.com/education/what-is-zero-trust-privilege, 2019.

Walid Rjaibi is Distinguished Engineer and Chief Technology Officer (CTO) for Data Security with IBM in Toronto, Canada. Prior to his current role, Walid was Research Staff Member in network security and cryptography with IBM Research in Zurich, Switzerland. Walid's work on Data Security has resulted 26 granted patents and several publications in journals and conference proceedings such as the IDUG solutions journal, the internation conference on security and cryptography (SECRYPT), the internation conference on data engineering (ICDE), and the internation conference on Very Large Databases (VLDB).


Mohammad Hammoudeh is the Head of the CfACS IoT Laboratory and a Reader in Future Networks and Security with the Department of Computing and Mathematics, Manchester Metropolitan University. He has been a researcher and publisher in the field of big sensory data mining and visualization. He is a highly proficient, experienced, and professionally certified cybersecurity professional, specializing in threat analysis, and information and network security management. His research interests include highly decentralized algorithms, communication, and cross-layered solutions to Internet of Things, and wireless sensor networks.

Towards Zero-Trust Database Security – Part 2

Walid Rjaibi, Mohammad Hammoudeh

1 Introduction

In Part One, we have explored the direct and indirect means through which the same data in a database system can be accessed and the challenges they pose to adhering to the basic tenets of zero-trust security. Here, we outline the solutions that are most suitable to address these challenges and enable enterprises to implement zero-trust database security without negatively impacting core database tenets such as query performance.

2 Separation of duties

Traditionally, database systems have been designed such that the Database Administrator (DBA) manages all aspects of the database, including security and auditing. Additionally, the DBA inherently had full access to all tables in the database. With the emergence of insider threats as a security concern equally important to external threats [1], this traditional model clearly hampers an organization's ability to fully implement zero-trust database security.

We contend that database systems must provide the capability to allow organizations to vest security administration and database administration into two non-overlapping roles so separation of duties can be enforced. Separation of duties also ensures that the DBA does not have implicit access to all the data in the database. This separation of duties enables organizations to better adhere to zero-trust security. It may also dictate the type of database system to adopt as not all database systems necessarily provide the capabilities to enforce separation of duties.

3 Data encryption

Indirect access is most dangerous as it completely bypasses all access control and auditing in the database system. A powerful countermeasure to protect against indirect access is data encryption as encrypted data is of no value to an attacker. However, data encryption for database systems comes in many forms and not all forms of encryption address the indirect access threats outlined. There are also performance implications that need to be taken into account when selecting a database encryption solution.

Fig. 1 contrasts the key database encryption options. Self-Encrypting Disks and file system encryption provide the broadest coverage (they encrypt entire disks or file systems), but they only protect against indirect access to storage media. Tablespace encryption, full database encryption and column encryption protect against indirect access to both storage media and file system. Column encryption, however, is intrusive to applications and negatively affects performance. Tablespace encryption may create a vulnerability when a DBA inadvertently moves data from an encrypted tablespace to an unencrypted one, or when data is held in temporary tablespaces. Therefore, full database encryption allows organizations to implement zero-trust security without having to compromise either on the database side or on the security side. The design of one such solution is discussed in detail in [2]. To give an example, consider a classical 3-tier banking application which stores client data in its backend database. To protect this data against indirect access, the application would enable full database encryption for its backend database. Using the solution discussed in [2], this can be achieved using SQL as follows:

CREATE DATABASE <db-name> ENCRYPT

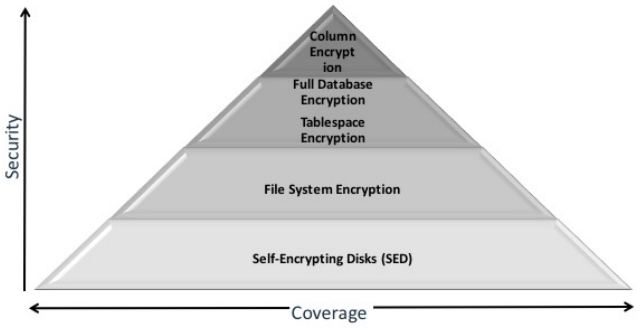


Fig. 1. Database encryption options.

4 Fine-grained access control

Fine-Grained Access Control (FGAC) refers to the ability to control access to database tables at the row level, column level, or cell level. This level of granularity ensures users are granted only the privileges they need and is paramount for mitigating the direct access scenarios outlined in Part One. However, database FGAC comes in many forms and not all forms adequately address the direct access threats. There are also usability implications that need to be taken into account when selecting a database FGAC solution.

Fig. 2 contrasts the database FGAC options. Database views [3] and application-based FGAC provide most flex-

ibility in terms of expressing FGAC rules, but the security they provide is not data-centric and can be bypassed. Label-Based Access Control (LBAC) [5] is a data-centric security model where the security policy is always enforced regardless of whether the table is accessed directly or indirectly through a view. However, LBAC lacks in flexibility when it comes to expressing security rules outside of the rigid No Read-Up and No Write-Down rules of Multilevel Security (MLS) [6]. Row permissions and column masks [4] combine the benefits of views and LABC. They are very suitable to implementing zero-trust security. To give an example, consider our banking application again. Suppose that client data is stored in a table called CLIENT. Further, suppose that the bank's security policy is such that only members of the TELLER role can see the full account number in table CLIENT. Anyone else can only see the last 4 digits. Using the solution discussed in [4], this can be achieved using SQL as shown below. The mask construct created is automatically evaluated by the database system each time the account number column is accessed and ensures the bank's security policy is enforced.

```
CREATE MASK ACCOUNT_ACCESS
ON CLIENT
FOR COLUMN account RETURN
CASE WHEN
  VERIFY_ROLE_FOR_USER (USER, 'TELLER') = 1
  THEN account
  ELSE 'XXXX-'|| SUBSTR(ACCOUNT,5,4)
END
ENABLE;
```

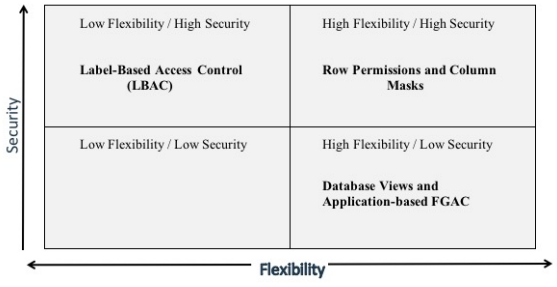


Fig. 2. Database FGAC options.

5 User Identity Propagation in Multitiered Environments

In multitiered database environments, the application interacts with the database system using a generic user ID. This model does not contribute to implementing zero-trust security because the database system does not see the actual end user identities. One major implication of this is diminished user accountability as the database audit log will only show a generic user ID with no references to the actual end users behind the application.

Some database systems provide the notion of *Application Context* to give applications the tools to propagate the end user identity to the database system where it can be used for auditing purposes [7]. In other solutions such as the Trusted Context concept introduced in [4], a more formal mechanism is used to allow the establishment of a trust relationship between the database system and the application and for the propagation of end user identities to the database system in a controlled and secure manner.

Strategies for implementing zero-trust database security must consider multitiered database environments to ensure that user accountability is maintained. This may in turn dictate the type of database system to adopt as not all database systems necessarily provide the capabilities to enable applications to propagate end user identities. To give an example, let's continue with our banking application. To ensure that the actual end user identities are propagated to the database, the application can leverage the trusted context concept introduced in [4]. This requires the following steps:

1. The administrator creates a trusted context object to define a trust relationship between the application and its backend database.
2. The application establishes a trusted connection with its backend database.
3. Before issuing any request to the database on behalf of an end user, the application switches the current user of the connection to the new user. This automatically propagates the end user identity to the database where it is used for all access control and auditing till the application switches user again.

6 Conditional Authorization

Traditional database authorization does not provide control around when a particular privilege can be exercised. One major use case where this model falls short is application bypass. An application administrator may choose to abuse the application credentials by accessing the database outside the scope of the application.

Some database systems provide a capability to allow organizations to require the database system to verify more attributes before allowing a user to exercise their privileges. For example, the Trusted Context concept introduced in [4] addresses application bypass by requiring the database system to authorize the application user ID only when additional attributes have been verified. Therefore, an application administrator who wishes to abuse the application credentials by accessing the database outside the scope of the application will find it hard to do so.

Strategies for implementing zero-trust database security must consider enforcing conditional authorization to protect against privilege abuse scenarios. This may also influence the choice of the database system to adopt as not all database systems necessarily support conditional authorization.

7 CONCLUSIONS

Databases contain enterprises most critical data and are the subject of attacks by both insiders and outsiders. Implementing zero-trust database security is therefore paramount to protect critical data. While user authentication, Transport Layer Security and auditing are standard practices and are usually implemented adequately by most organizations, the indirect and direct threats outlined in this paper require careful thinking including the choice of the database system to adopt. Table 1 summarizes the indirect and direct threats we outlined together with the security best practices to address them and enable adherence to zero-trust security.

TABLE 1
IMPLEMENTING ZERO-TRUST DATABASE SECURITY

Threat	Threat type	Security best practice
File system access	Indirect	Full database encryption
Storage media access	Indirect	Full database encryption
Privilege abuse	Direct	- Separation of duties - Fine-Grained Access Control (FGAC)
Application bypass	Direct	Conditional authorization
Loss of end user identity in multitiered environments	Direct	User identity propagation

REFERENCES

[1] Verizon, https://www.knowbe4.com/hubfs/rp_DBIR_2017_Report_execsummary_en_xg.pdf, 2017.

[2] W. Rjaibi, "Holistic Database Encryption", *Proc. International Conference on Security and Cryptography*, 2018.

[3] R. Elmasri, S. Navathe, *Fundamentals of Database Systems 6th*. Addison-Wesley, 2010.

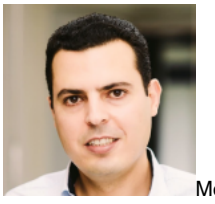
[4] W. Rjaibi, M. Hammoudeh, " Fine-Grained Database Authorization and User Identity Propagation in Multitiered Environments", *IEEE Trans. On Knowledge and Data Engineering*, submitted for publication (Pending evaluation), 2019.

[5] W. Rjaibi, P. Bird, "A Multi-Purpose Implementation of Mandatory Access Control in Relational Database Management Systems", *Proc. International Conference on Very Large Data Bases*, 2004.

[6] W. Rjaibi, "An introduction to multilevel secure relational database management systems", *Proc. The conference of the Centre for Advanced Studies on Collaborative research (CASCON)*, 2004.

[7] Oracle, "Defense-in-Depth Database Security for On-Premises and Cloud Databases", https://www.oracle.com/technetwork/database/security/security-compliance-wp-12c-1896112.pdf., 2019.

Walid Rjaibi is Distinguished Engineer and Chief Technology Officer (CTO) for Data Security with IBM in Toronto, Canada. Prior to his current role, Walid was Research Staff Member in network security and cryptography with IBM Research in Zurich, Switzerland. Walid's work on Data Security has resulted 26 granted patents and several publications in journals and conference proceedings such as the IDUG solutions journal, the international conference on security and cryptography (SECRYPT), the international conference on data engineering (ICDE), and the international conference on Very Large Databases (VLDB).



Mohammad Hammoudeh is the Head of the CfACS IoT Laboratory and a Reader in Future Networks and Security with the Department of Computing and Mathematics, Manchester Metropolitan University. He has been a researcher and publisher in the field of big sensory data mining and visualization. He is a highly proficient, experienced, and professionally certified cybersecurity professional, specializing in threat analysis, and information and network security management. His research interests include highly decentralized algorithms, communication, and cross-layered solutions to Internet of Things, and wireless sensor networks.

Appendix C: Mandatory Access Control Portfolio

Table C.1 – Research Papers

ID	Publication	Key Contributions
1	**A Multi-Purpose Implementation of Mandatory Access Control in Relational Database Management Systems** *Very Large Databases (VLDB) Conference*	- Design of a mandatory access control solution for database systems which addresses the limitations of traditional Multilevel Security (MLS). - Enable organizations to adhere to zero-trust security. - Implementation of the solution in IBM DB2 for Linux, Unix and Windows, and Informix.
2	**Inter-Node Relationship Labeling: A Fine-Grained XML Access Control Implementation Using Generic Security Labels** *International conference on security and cryptography (SECRYPT)*	- Design of a solution which improves over traditional node-based XML access control approaches, by considering inter-node relationships as the control granularity. - Enable databases to extend fine-grained authorizations to XML columns in database tables. - Enable organizations to meet privacy requirements and adhere to zero-trust security.
3	**An Introduction to Multilevel Secure Relational Database Management Systems** *International Conference on Computer Science and Software Engineering*	Survey and critique of traditional implementations of mandatory access control in database systems (i.e., MLS).

A Multi-Purpose Implementation of Mandatory Access Control in Relational Database Management Systems

Walid Rjaibi Paul Bird

IBM Toronto Software Laboratory
8200 Warden Avenue
Markham, Ontario
Canada
{wrjaibi, pbird}@ca.ibm.com

Abstract

Mandatory Access Control (MAC) implementations in Relational Database Management Systems (RDBMS) have focused solely on Multilevel Security (MLS). MLS has posed a number of challenging problems to the database research community, and there has been an abundance of research work to address those problems. Unfortunately, the use of MLS RDBMS has been restricted to a few government organizations where MLS is of paramount importance such as the intelligence community and the Department of Defense. The implication of this is that the investment of building an MLS RDBMS cannot be leveraged to serve the needs of application domains where there is a desire to control access to objects based on the label associated with that object and the label associated with the subject accessing that object, but where the label access rules and the label structure do not necessarily match the MLS two security rules and the MLS label structure. This paper introduces a flexible and generic implementation of MAC in RDBMS that can be used to address the requirements from a variety of application domains, as well as to allow an RDBMS to efficiently take part in an end-to-end MAC enterprise solution. The paper also discusses the extensions made to the SQL compiler component of an RDBMS to incorporate the label access rules in the access plan it generates for an SQL query, and to prevent unauthorized leakage of data that could occur as a result of traditional optimization techniques performed by SQL compilers.

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

1 Introduction

Mandatory Access Control (MAC) is a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity[8]. A well-known implementation of MAC is Multilevel Security (MLS), which, traditionally, has been available mainly on computer and software systems deployed at highly sensitive government organizations such as the intelligence community or the U.S. Department of Defense. The Basic model of MLS was first introduced by Bell and LaPadula[9]. The model is stated in terms of objects and subjects. An object is a passive entity such as a data file, a record, or a field within a record. A subject is an active process that can request access to objects. Every object is assigned a classification, and every subject a clearance. Classifications and clearances are collectively referred to as labels. A label is a piece of information that consists of two components: A hierarchical component and a set of unordered compartments. The hierarchical component specifies the sensitivity of the data. For example, a military organization might define levels Top Secret, Secret, Confidential and Unclassified. The compartments component is nonhierarchical. Compartments are used to identify areas that describe the sensitivity or category of the labeled data. For example, a military organization might define compartments NATO, Nuclear and Army. Labels are partially ordered in a lattice as follows: Given two labels L_1 and L_2, $L_1 >= L_2$ if and only if the hierarchical component of L_1 is greater than or equal to that of L_2, and the

compartment component of L1 includes the compartment component of L_2. L_1 is said to *dominate* L_2. MLS imposes the following two restrictions on all data accesses:

- The Simple Security Property or "No Read Up": A subject is allowed a read access to an object if and only if the subject's label dominates the object's label.
- The *-Property (pronounced the star property) or "No Write Down": A subject is allowed a write access to an object if and only if the object's label dominates the subject's label.

1.1 Problem Statement

MAC implementations in Relational Database Management Systems (RDBMS) have focused solely on MLS. MLS has posed a number of challenging problems to the database research community, and there has been an abundance of research work to address those problems. There has also been three commercial MLS RDBMS offerings, namely, Trusted Oracle[16], Informix OnLine/Secure[17], and Sybase Secure SQL Server[20]. Unfortunately, the use of MLS RDBMS has been restricted to a few government organizations where MLS is of paramount importance such as the intelligence community and the Department of Defense. In fact, very few commercial organizations need such type of security. The implication of this is that the investment of building an MLS RDBMS cannot be leveraged to serve the needs of application domains where there is a desire to control access to objects based on the label associated with that object and the label associated with the subject accessing that object, but where the label access rules and the label structure do not necessarily match the MLS two security rules and the MLS label structure (*i.e.,* a hierarchical component and a set of unordered compartments). The question that begs to be asked is therefore the following: Do such application domains exist and, if so, what are they?

We contend that the answer to that question is an unequivocal yes. Privacy[19] is one example of such application domain. Generally, a privacy policy indicates for which purposes an information is collected, whether or not it will be communicated to others, and for how long that information is retained before it is discarded. For example, a user cannot access a customer record for the purpose of sending that customer marketing information if that customer did not agree to receive such information. Access to privacy-sensitive data can be regarded as analogous to access to MLS data in the sense that in both cases there is a tag associated with the object being accessed and the subject accessing that object. The tag represents a "purpose" in the case of the former and represents a "security label" in the case of the latter. Unfortunately, a MAC implementation in an RDBMS that strictly implements MLS fails to address privacy requirements for the following two main reasons. First, MLS labels include a hierarchical component that is not applicable in the case of privacy. Next, the MLS security properties do not apply in the context of privacy. For example, to read an object in MLS, the subject's compartment component must include that object's compartment component (the simple security property). In privacy, the rule is exactly the opposite. That is, if an object is tagged with the purposes *marketing* and *purchase*, then a user accessing that object for the purpose of sending marketing information must be allowed to access that object.

Another application domain is private banking. In private banking, country laws and regulations often require to limit the amount of data that can be viewed by a bank employee. For example, Swiss banking laws do not allow a Swiss bank employee located in Toronto to access account information for customers based in Switzerland. Typically, banking applications code this fine-grained access control in the application itself, as opposed to delegating this task to the RDBMS. Unfortunately, this *application-aware* approach has made enterprise security policies a laborious and complex task. It also has the drawback of exposing the security policies to the application programmers. If each customer account is tagged with a label indicating the geographical location of the customer and if each bank employee can be assigned a label that also indicates the geographical location of that employee (for example, based on the system security context established when that employee logs on to the database), then an RDBMS that implements a form of MAC where the database administrator could define the label structure and the label access rules could relieve the applications from implementing such fine-grained access control policies.

Moreover, the ever increasing enterprise demands for more security has led to the emergence of label security products that provide the ability to set up and control access based upon labels throughout an entire network from end-to-end. For example, such label security products have the ability to control the network to decide whether or not a particular labeled data row can be transmitted on a particular channel or be delivered to a particular workstation on that network. An important advantage of such label security products is their ability to offer a centrally managed tool for defining label access policies and for assigning access labels to users as well as to other entities on the network. Traditional implementations of MAC in RDBMS (*i.e.,* MLS) do not offer the required flexibility to efficiently integrate with such label security products and to provide pervasive system coverage using a unified and centrally managed label access policy.

Therefore, there is a need for a flexible and generic implementation of MAC in RDBMS that can be used to address the requirements from a variety of application domains, including those of MLS, and to efficiently take part in an end-to-end MAC enterprise solution.

1.2 Contributions

The contributions made in this paper can be summarized as follows:

1. A methodology to define labels and to set up a database table such that access to a row in that table is based upon the label associated with that row and the label associated with the user accessing that row. More specifically, the methodology introduces a number of extensions to SQL that would allow a database administrator to:
   - Define label types
   - Define label access rules and exceptions to them
   - Assign labels and exceptions to database users
   - Attach a label type and a set of label access rules to a database table
2. Extensions to the SQL compiler component of an RDBMS to:
   - Incorporate the label access rules in the access plan it generates for an SQL query
   - Prevent unauthorized leakage of data that could occur as a result of traditional optimization techniques performed by SQL compilers
3. Extensions to the runtime processor component of an RDBMS to enforce label access rules
4. A method to allow an RDBMS to efficiently take part in an end-to-end MAC enterprise solution

1.3 Synopsis

Section 2 gives a brief survey of MAC implementations in RDBMS. Section 3 introduces our methodology for defining labels and for setting up a database table such that access to a row in that table is based upon the label associated with that row and the label associated with the user accessing that row. Section 4 presents our extensions to the SQL compiler component of an RDBMS to incorporate the label access rules in the access plan it generates for an SQL query, and to prevent unauthorized leakage of data that could occur as a result of traditional optimization techniques performed by SQL compilers. Section 5 describes our extensions to the methodology introduced in section 3 in order to allow an RDBMS to efficiently take part in an end-to-end MAC enterprise solution. Lastly, section 6 summarizes our results and discusses future work.

2 Related Work

MAC implementations in Relational Database Management Systems have focused solely on MLS, which is of paramount importance to a few government organizations such as the intelligence community or the Department of Defense. In fact, there has been an abundance of research within the last two decades or so in the area of multilevel secure relational databases. The results of such research can be divided into three broad areas as follows.

2.1 Multilevel Secure Relational Database Models

The Sea View model[1] was the pioneering formal multilevel secure relational database designed to provide mandatory access control. It extended the concept of a database relation to include the security labels. A relation that is extended with the security labels is called a multilevel relation. The Sea View model also coined the concept of *polyinstantiation*, which refers to the simultaneous existence of multiple tuples with the same primary key, where such tuples are distinguished by their security labels. In order to avoid covert channels, subjects with different security labels are allowed to operate on the same database relation through the use of polyinstantiation[1]. The Jajodia-Sandhu model[2] was derived from the Sea View model. It was shown in [3] that the Sea View model can result in the proliferation of tuples on updates and the Jajodia-Sandhu model addresses this drawback. The Smith-Winslett model[4] was the first model to extensively address the semantics of an MLS database. The MLR model[5] is based on the Jajodia-Sandhu model, and also integrates the belief-based semantics of the Smith-Winslett model. It was shown in [7] that all the aforementioned models can present users with some information that is difficult to interpret. The BCMLS model[6] addresses those concerns by including the semantics of an unambiguous interpretation of all data presented to the users.

2.2 Multilevel Secure RDBMS Architectures

Multilevel secure RDBMS architectures schemes can be divided into two general categories: The Trusted Subject architecture and the Woods Hole architectures.

The Trusted Subject architecture[10] is a scheme that contains a trusted RDBMS and a trusted operating system. The RDBMS is custom-developed to include all the required security rules, but uses the associated trusted operating system to make actual disk data accesses. A benefit of this scheme is that the

RDBMS has access to all levels of data at the same time, which minimizes retrieval and update processing. However, this scheme results in a special purpose RDBMS that requires a large amount of trusted code to be developed and verified along with the normal RDBMS features.

The Woods Hole architectures assume that an untrusted off-the-shelf RDBMS is used to access data and that trusted code is developed around that RDBMS to provide an overall secure RDBMS. They can be divided into two main categories: The kernelized architectures and the distributed architectures[10, 11].

The kernelized architecture scheme uses a trusted operating system and multiple copies of the RDBMS, where each copy is associated with some trusted front-end. Each pair (trusted front-end, RDBMS) is associated with a particular security level. The trusted operating system ensures that data at different security levels is stored separately, and that each copy of the RDBMS gets access to data that is authorized for its associated security level. A benefit of this scheme is that data at different security levels is isolated in the database, which allows for higher level assurance. However, this scheme results in an additional overhead as the trusted operating system needs to separate data at different security levels when it is added to the database and might also need to combine data from different security levels when data is retrieved by an RDBMS copy that is associated with a high security level.

The distributed architecture scheme uses multiple copies of the trusted front-end and RDBMS, each associated with its own database storage. In this architecture scheme, an RDBMS at security level l contains a replica of every data item that a subject at level l can access. Thus, when data is retrieved, the RDBMS retrieves it only from its own database. Another benefit of this architecture is that data is physically separated into separate hardware databases. However, this scheme results in an additional overhead when data is updated as the various replicas need to be kept in sync.

2.3 Multilevel Secure Transaction Processing

Although the two MLS security properties described above prevent direct legal flow of information from a security level to another nondominated security level, they are not sufficient to ensure that security is not compromised since it could be possible for leakage of information to occur through indirect means via covert channels. A covert channel can easily be established with conventional concurrency control algorithms such as two-phase locking (2PL) and timestamp ordering (TO). In both 2PL and TO algorithms, whenever there is contention for the same data item by transactions executing at different security levels, a lower level transaction may be either delayed or suspended to ensure correct execution. In such a scenario, two colluding transactions executing at high and low security levels can establish an information flow channel from a high security level to a low security level by accessing the selected data item according to some agreed-upon code[12].

Considerable effort has been devoted to the development of efficient, secure algorithms for the major schemes of RDBMS architectures described above. In [13], Keefe *et al.* present a formal framework for secure concurrency control in multilevel databases. Lamport[14] offers solutions to the secure readers/writers problem. While these solutions are secure, they do not yield serializable schedules when applied to transactions, and they suffer from the problem of starvation, *i.e.,* transactions that are reading data items at a low security level may be delayed indefinitely[18]. In [15], Ammann and Jajodia offer two timestamp-based algorithms that yield serializable schedules, but both suffer from starvation. On the commercial secure RDBMS side, both Trusted Oracle RDBMS[16] and Informix OnLine/Secure RDBMS[17] offer concurrency control solutions that are free from covert channels.

3 Methodology for Setting up MAC in an RDBMS

The methodology we propose allows a database administrator to define labels and to set up a database table such that access to a row in that table is based upon the label associated with that row and the label associated with the user accessing that row. More specifically, the methodology allows the database administrator to:

- Define label types
- Define label access rules and exceptions to them
- Assign labels and exceptions to database users
- Attach a label type and a set of label access rules to a database table

We now introduce our extensions to SQL to implement this methodology. The goal of this exercise is not to describe the blueprint for the implementation. Rather, we will focus on the new SQL concepts that must be implemented to support such methodology. Also, we have chosen not to overload the paper with the details of the exact syntax of the SQL extensions proposed, as we believe that such level of details is more appropriate for a standardization proposal to the SQL standard committee. However, we will illustrate the syntax and the concepts introduced via examples.

3.1 Label Component

A label component is a database entity that can be created, altered and dropped. It is introduced as a

building block for labels (*i.e.,* a label is composed of one or more label components). The label component definition specifies the set of valid elements for that label component. This set of elements can be either ordered or unordered (the default). In an ordered set, the order in which the elements appear is important: The rank of the first element is higher than the rank of the second element, the rank of the second element is higher than the rank of the third element, and so on. To allow database administrators to create, alter and drop label components, we introduce the CREATE, ALTER and DROP label component SQL statements. The CREATE LABEL COMPONENT SQL statement creates a label component that can be used to define a label type. The ALTER LABEL COMPONENT SQL statement permits to add or drop an element to/from a label component. The DROP LABEL COMPONENT SQL statement drops a label component.

Example 1

The following SQL statement creates a label component called level and specifies the set of valid values for this label component.

```
CREATE LABEL COMPONENT level
OF TYPE varchar(15)
USING ORDERED SET
{"TOP SECRET", "SECRET", "CLASSIFIED"}
```

The following SQL statement creates a label component called compartments and specifies the set of valid values for this label component. Note that the set specified is unordered.

```
CREATE LABEL COMPONENT
compartments OF TYPE varchar(15)
USING SET
{"NATO", "NUCLEAR", "ARMY"}
```

The following SQL statement adds a new element to the level component and specifies the rank of this new element within the ordered set.

```
ALTER LABEL COMPONENT level
ADD ELEMENT "UNCLASSIFIED"
AFTER "CLASSIFIED"
```

The following SQL statement drops the level component.

```
DROP LABEL COMPONENT level
```

3.2 Label Type

The relationship between a label and a label type is analogous to the relationship between a data row and a table schema. As the table schema defines the set of columns that make up a data row, so the label type defines the set of label components that make up a label. To allow database administrators to create, alter and drop label types, we introduce the CREATE, ALTER and DROP label type SQL statements. The CREATE LABEL TYPE creates a label type by specifying the label components that make up such label type. The ALTER LABEL TYPE alters the definition of a label type by adding or dropping a label component to/from that label type. The DROP LABEL TYPE SQL statement drops a label type.

Example 2

The following SQL statement creates a label type called MLS and specifies its label components. Note the keyword MULTIVALUED next to the compartments component. This indicates that the compartments component can have more than a single value at a time. This keyword can only be specified for label components based on an unordered set (section 3.4 explains the reason behind this choice).

```
CREATE LABEL TYPE MLS
COMPONENTS level,
compartments MULTIVALUED
```

The following SQL statement drops the level component from label type MLS.

```
ALTER LABEL TYPE MLS DROP level
```

The following SQL statement drops the MLS label type.

```
DROP LABEL TYPE MLS
```

3.3 Access Labels and Row Labels

We distinguish two types of labels: *Access labels* and *row labels*. Access labels are created and assigned to database users, which, in conjunction with the label access rules (section 3.4), determine which labeled rows these users have access to. To allow database administrators to create, drop, grant and revoke access labels, we introduce the CREATE, DROP, GRANT and REVOKE access label SQL statements. The CREATE ACCESS LABEL SQL statement creates an access label based on an existing label type. The GRANT ACCESS LABEL SQL statement grants an access label to a database user. The REVOKE ACCESS LABEL SQL statement revokes an access label from a database user. The DROP ACCESS LABEL SQL statement drops an access label and revokes it from any database user to whom it has been granted.

Example 3

The following SQL statement creates an access label.

CREATE ACCESS LABEL L_1
OF LABEL TYPE MLS
level "SECRET", compartments "NATO"

The following SQL statement grants access label L_1 to database user Joe.

GRANT ACCESS LABEL L_1
TO USER Joe

The following SQL statement revokes access label L_1 from database user Joe.

REVOKE ACCESS LABEL L_1
FROM USER Joe

The following SQL statement drops access label L_1.

DROP ACCESS LABEL L_1

A row label labels a data row in a database table. To allow database users to provide a row label when inserting or updating a row in a database table, we introduce the ROWLABEL function. ROWLABEL is a means of providing the label value of a data row.

Example 4

The following INSERT SQL statement shows how the row label can be provided using the ROWLABEL function. The statement inserts a row into a database table called T1 having two columns A and B both of type integer. We assume that rows in table T1 are labeled with a label of label type MLS defined above.

INSERT INTO T1 VALUES
(ROWLABEL("SECRET", "NATO"), 1, 2)

The following SQL statement shows how the ROWLABEL function can be used to update the level component of the row label for the row inserted above.

UPDATE T1 SET
ROWLABEL(level) = "TOP SECRET"
WHERE A = 1 AND B = 2

3.4 Label Access Policy

A label access policy defines the label access rules that the RDBMS evaluates to determine whether or not a database user is allowed access to a labeled data row in a database table. Access rules can be divided into two categories: Read access rules and write access rules. Read access rules are applied by the RDBMS when a user attempts to read a labeled data row (e.g., a SELECT statement). The RDBMS applies the write access rules when a user attempts to insert, update or delete a labeled data row. In both cases, an access rule is a predicate that puts together the same component from an access label and a row label and an operator as follows:

Access Label *component-name*
<operator>
Row Label *component-name*

The type of operator allowed depends on the label component. For label components based on an ordered set, the operator can be any of the relational operators $\{=, <=, <, >, >=, !=\}$. For label components based on an unordered set, the operator must be one of the set operators {IN, INTERSECT}. Recall from section 3.2 that a label component based on an unordered set can be multivalued. That is, it can contain more than a single value at a time. Thus, when comparing multivalued label components we are actually comparing data sets. This is the reason why the operators supported are set operators, *i.e.,* inclusion and intersection. Obviously, certain RDBMS could choose to support additional operators but we contend that the ones given above would be the most commonly used. To allow database administrators to create, alter and drop label policies, we introduce the CREATE, ALTER and DROP label policy SQL statements. The CREATE LABEL POLICY SQL statement creates a label access policy for a given label type by specifying one or more read access rules and one or more write access rules. The ALTER LABEL POLICY SQL statement permits the addition or dropping an access rule to/from a label access policy. The DROP LABEL SQL statement drops a label access policy.

Example 5

The following SQL statement creates a label access policy that implements the two MLS properties introduced in section 1 above (*i.e.,* "No Read Up" and "No Write Down").

CREATE LABEL POLICY mls-policy
LABEL TYPE MLS
READ ACCESS RULE rule1
  ACCESS LABEL level >= ROW LABEL level
READ ACCESS RULE rule2
  ROW LABEL compartments IN
  ACCESS LABEL compartments
WRITE ACCESS RULE rule1
  ACCESS LABEL level <= ROW LABEL level

WRITE ACCESS RULE rule2
ACCESS LABEL compartments IN
ROW LABEL compartments

The following SQL statement drops read access rule rule2 from label access policy mls-policy.

ALTER LABEL POLICY mls-policy
DROP READ ACCESS RULE rule2

The following SQL statement drops label access policy mls-policy.

DROP LABEL POLICY mls-policy

3.5 Exceptions

Exceptions are introduced to provide the flexibility for some database users to bypass one or more access rules. For example, in an MLS context, it is often the case that some special users are allowed to write information to lower security levels even though this is in contradiction with the *-security property. Thus, exceptions are introduced to allow the database administrator to grant a database user an exception to bypass one or more access rules in a particular label access policy. To allow database administrators to grant and revoke exceptions, we introduce the GRANT and REVOKE exception SQL statements. The GRANT EXCEPTION SQL statement grants a database user an exception to bypass one or more access rules in a label access policy. The REVOKE EXCEPTION SQL statement revokes a previously granted exception from a database user.

Example 6

The following SQL statement grants an exception to database user Joe so that he can bypass the write access rules in label access policy mls-policy.

GRANT EXCEPTION
ON WRITE ACCESS RULE rule1, rule2
FROM LABEL POLICY mls-policy
TO USER Joe

The following SQL statement revokes the above exception from user Joe.

REVOKE EXCEPTION
ON WRITE ACCESS RULE rule1, rule2
FROM LABEL POLICY mls-policy
FROM USER Joe

3.6 Labeled Tables

A labeled table is a database table that contains labeled data rows. When the database administrator creates a labeled table he/she specifies the label type and the label access policy to be used for that table. The label type determines the structure of the label to be used to label the table's data rows and the label access policy determines the access rules to be used for enforcing access to that labeled table. To allow database administrators to create labeled tables, we extend the CREATE TABLE SQL statement by a new optional clause to specify the label type and the label access policy.

Example 7

The following SQL statement creates a database table T1 and specifies the label type and the label access policy. Note that in our examples so far we have used MLS-like label types and label access policies because they are well understood by the database research community. But it is obvious that one can follow the methodology given in this paper to define any label type and any label access policy, and attach them to a database table.

CREATE TABLE T1 (A integer, B integer)
LABEL TYPE MLS
LABEL POLICY mls-policy

When creating such table, the RDBMS internally adds a third column to store the label associated with each row in this table. The choice of the column's type depends on the label type. For example, if the label type is made up of a single component of type, say varchar(15), then the column's type would be varchar(15). If the label type is made up of more than a single column then the column's type must be an Abstract Data Type (ADT). ADTs have been introduced in SQL'99[21] and are supported by most commercial RDBMS. Alternatively, the RDBMS could choose not use an ADT and store the different label components in separate columns.

4 Extensions to the SQL Compiler Component in an RDBMS

When a labeled table is accessed, the RDBMS needs to enforce two levels of access control. The first level is the traditional Discretionary Access Control (DAC) which is implemented by all commercial RDBMS[21]. That is, the RDBMS verifies whether the user attempting to access the table has been granted the required privilege to perform the requested operation on that table. A discussion of this level of access control is beyond the scope of this paper. The second level is MAC. That is, for each data row accessed, the RDBMS verifies whether the user is allowed access to that row based on the label associated with the row and the user's access label.

4.1 Enforcing MAC on Labeled Tables

There are two possible ways that MAC can be enforced when a labeled table is accessed. The first possibility is for the SQL compiler to modify any query that refers to a labeled table in order to incorporate the access rules from the label access policy associated with that table in the form of regular predicates. Next, the SQL compiler compiles the modified query and generates an access plan for the query in the normal fashion. The main advantage of such an approach is its simplicity. However, it has a major drawback: The access plan generated for a query that refers to a labeled table cannot be reused by other users because it is dependent on the access label of the user who issued the query. Note that some commercial RDBMS cache the access plan generated for an SQL query so that it can be reused the next time the SQL query is submitted. This has some performance benefits as it eliminates the need to recompile the query. Another drawback of this approach is that it could result in unauthorized leakage of data if special care is not taken by the SQL compiler. This will be detailed further in section 4.2.

The second possibility is to not modify a query that refers to a labeled table. Rather, the SQL compiler inserts logic into the access plan that implements the access rules from the label access policy associated with any labeled table referred to in the query. Thus, when the access plan is executed, the access rules from the label access policy associated with a labeled table are evaluated for each data row when that labeled table is accessed. The general processing algorithm to be inserted in the access plan for a labeled table is as follows.

```
Begin
  Fetch the user's access label (e.g., from a
  system catalog table)
  if (SELECT access)
  {
     for each row accessed
     {
        if (read access rules do not permit access)
        {
           Skip row
        }
     }
  }
  else
  {
        // INSERT, UPDATE, or DELETE access
        for each row
        {
           if (INSERT or UPDATE)
           {
              if (the row label provided is not valid with
              respect to the label type associated with
              the labeled table)
                 Reject INSERT or UPDATE
           }
           if (write access rules do not permit access)
              Reject INSERT, UPDATE or DELETE
        }
  }
End
```

This second approach addresses the two shortcomings of the previous approach (i.e., query modification). That is, it allows the cached access plan to be reused because the access label of the user who issued the query is acquired at runtime, and it is more secure as it will be demonstrated in section 4.2.

4.2 Predicates Evaluation Sequence

SQL compilers have traditionally been guided by performance reasons in selecting the order in which the predicates contained in a query are evaluated. For example, more selective predicates are often evaluated first to narrow down the set of rows to be passed on to a subsequent join because join operations are costly. If the method chosen to enforce MAC on a labeled table is based on query modification to incorporate the access rules in the form of regular predicates, then special care must be taken in selecting the order in which the predicates on that table are evaluated to avoid unauthorized leakage of labeled data rows. For example, suppose that a query has a predicate on a labeled table that involves a User-Defined Function (UDF). Further suppose that this UDF takes the whole data row as an input parameter and that the UDF source code makes a copy of the data row outside the database (or sends it as an e-mail to some destination). Now, suppose that some data row R cannot be returned to the user who issued the query because this would violate the access rules from the label access policy associated with this labeled table. If the predicate involving the UDF is evaluated prior to evaluating the predicates that implement the access rules then data row R will be consumed by the UDF and consequently leaked to an unauthorized user.

If the RDBMS chooses the query modification method to enforce MAC on a labeled table, then it must ensure that the predicates that implement the access rules are evaluated before any other predicate so that no labeled row leakage could occur. The alternative approach that is not based on query modification evaluates the access rules immediately after the row is accessed, and before any predicate is evaluated. It is therefore more secure than the query modification approach. It also allows the SQL compiler to continue to select the order in which predicates are evaluated in the usual way.

4.3 Index-Only Access Methods

When selecting an access plan, SQL compilers choose between three methods of accessing the data in a database table: Scanning the entire table sequentially, locating specific table rows by first accessing an index on the table, or accessing just an index on the table if all the required columns are part of the index key. This latter method is known as index-only access. SQL compilers usually rely on the statistics available about the table and the indices to choose between those three access methods. If an index only plan is selected then the label column is not available and therefore the access rules from the label access policy associated with the table cannot be evaluated. MLS RDBMS extended the primary key on an MLS relation with the security label column in order to allow the simultaneous existence of multiple tuples with the same (non extended) primary key (*i.e.,* polyinstantiation)[1]. We borrow this idea from the MLS work to extend every index created on a labeled table (including the primary key) with the row label column(s). This would allow SQL compilers to continue to choose index only access methods when this is appropriate, and for the access rules from the label access policy associated with the table on which the index is created to be evaluated.

5 Methodology for an End-to-end MAC Enterprise Solution

The ever-increasing enterprise demands for more security has led to the emergence of label security products that provide the ability to set up and control access based upon labels throughout an entire network from end to end. For example, such label security products have the ability to control the network to decide whether or not a particular labeled data row can be transmitted on a particular channel or be delivered to a particular workstation on that network. Cryptek[22] is an example of such a label security product. An important advantage of such label security products is their ability to offer a centrally managed tool for defining label access policies and for assigning access labels to users as well as to other entities on the network. We contend that a MAC implementation in RDBMS should offer the flexibility to integrate with a label security product for the following reasons:

1. Eliminate the need for the system administrator to define the label access rules in more than a single location (*i.e.,* both in the RDBMS and in the label security product)

2. Eliminate the need for the system administrator to assign access labels to users in more than a single location

3. Allow the access to a labeled data row in the database to be based on more sophisticated label access rules that a particular implementation of MAC in an RDBMS may not allow to express

We will now show how the methodology described earlier in this paper could be extended to allow an RDBMS to take part in such an end-to-end MAC scheme by providing the flexibility to integrate with a label security product.

5.1 Integration Approach

Recall from section 3.6 that we have extended the CREATE TABLE SQL statement with an optional clause to specify the label type and the label access policy. We further extend this SQL statement such that the LABEL POLICY clause could either specify the name of a label access policy defined within the RDBMS, or a label access policy defined externally to the RDBMS (*i.e.,* within a label security product). The keyword EXTERNAL is introduced to support this latter possibility as shown below.

```
CREATE TABLE T1 (A integer, B integer)
LABEL TYPE some-label-type
LABEL POLICY EXTERNAL
```

When a data row in such a table is accessed, the RDBMS needs to supply the ID of the user making the access together with the data row label and the table name to the label security product through a well-defined interface. The label security product evaluates the label access rules based on the information received from the RDBMS and returns a response to the RDBMS through that same interface. The response could be a Boolean flag indicating whether or not the access should be allowed.

The SQL compiler will now need to take into account where the label access rules are defined when inserting logic into an access plan to enforce MAC on a labeled table. Thus, a more general description of the algorithm to be inserted in the access plan for a labeled table is as follows.

```
Begin
  if (policy defined within RDBMS)
  {
     Fetch the user's access label (e.g., from a
     system catalog table)
  }
  if (SELECT access)
  {
     for each row accessed
     {
        if (policy defined within RDBMS)
        {
           if (read access rules do not permit access)
           {
              Skip row
```

```
            }
        }
        else
        {
            response = callLabelSecurityProduct(userid,
                        rowlabel, table-name)
            if (response is No)
            {
                Skip row
            }
        }
    }
}
else
{
        // INSERT, UPDATE, or DELETE access
        for each row
        {
            if (INSERT or UPDATE)
            {
                if (the row label provided is not valid with
                respect to the label type associated with
                the labeled table)
                    Reject INSERT or UPDATE
            }
            if (policy defined within RDBMS)
            {
                if (write access rules do not permit access)
                    Reject INSERT, UPDATE or DELETE
            }
            else
            {
                response = callLabelSecurityProduct
                            (userid, rowlabel, table-name)
                if (response is No)
                {
                    Reject INSERT, UPDATE or DELETE
                }
            }
        }
}
End
```

Clearly, the calls to the label security product, which is external to the RDBMS, would cause a performance degradation. In the next section, we will show how this performance degradation could be minimized.

5.2 Performance Improvement

To minimize the performance degradation that could result from the calls to the label security product, a caching technique could be used. Before making the call to the label security product, the RDBMS would first check the cache to see if a similar call was made earlier, and if so fetches the response directly from the cache. The cache structure could look as follows.

Userid	RowLabel	Table	Access	Resp.
Joe	L	T	Read	Yes
Bob	L'	T	Write	No

Table 1: Label security product responses cache

To ensure that the cache entries are always valid, the label security product must signal to the RDBMS through a well-defined interface any changes to the label access rules associated with a database table, or to the access labels assigned to a database user. When such a signal is received, the RDBMS invalidates the cache entries that are affected by the change in label access rules or user access labels. For example, if the label access rules associated with table T have changed, then all cache entries for table T must be invalidated. Similarly, if the access label for user Joe has changed or has been revoked, then all cache entries for user Joe must be invalidated.

6 Conclusion and Future Directions

This paper has introduced a flexible and generic implementation of MAC in RDBMS that can be used to address the requirements from a variety of application domains, as well as to allow an RDBMS to efficiently take part in an end-to-end MAC enterprise solution. This implementation differs from traditional MAC implementations in RDBMS, which have focused solely on MLS, and thus cannot be leveraged to serve the needs of application domains where there is a desire to control access to objects based on the label associated with that object and the label associated with the subject accessing that object, but where the label access rules and the label structure do not necessarily match the MLS two security rules and the MLS label structure (*i.e.,* a hierarchical component and a set of unordered compartments). Moreover, such implementations do not offer the flexibility to integrate with an external label security product and therefore cannot take part in an end-to-end MAC enterprise solution.

There are a number of additional problems related to implementing a generic MAC solution in an RDBMS that have not been addressed in this paper. These will be the subject of our future work. For example, triggers could cause labeled data rows to flow from a labeled table to a nonlabeled table if the subject of a trigger is a labeled table but the target of that trigger is a nonlabeled table. Without proper flow control measures, triggers could cause unauthorized leakage of information to occur. Also, there needs to be a mechanism to accommodate views based on labeled tables. For example, if a view is based on a join between two labeled tables how would the row label of a join result row be selected. Should the RDBMS make the decision about how to combine labels? or should the RDBMS offer the flexibility that would allow database administrators to provide the rules for combining two labels from the same label type?

Acknowledgements

Some of the ideas expressed in this paper were generated when the first author was a Research Staff Member at the IBM Zurich Research Lab (ZRL). The first author would like to thank Dr. Michael Waidner, manager Network Security & Cryptography, for giving him the opportunity to start up the database security research activity at ZRL. The first author would also like to thank his wife Hue Phan Dam for her valuable comments on an earlier version of this paper and for her help with the examples.

Trademarks

Disclaimer

The views expressed in this paper are those of the authors and not necessarily of IBM Canada Ltd. or IBM Corporation.

References

[1] D. E. Denning. The Sea View Security Model. In *Proc. of the IEEE Symposium on Security and Privacy*, Oakland, California, USA, 1988.

[2] S. Jajodia, R. Sandhu. Toward a Multilevel Secure Relational Data Model. In *Proc. of ACM SIGMOD*, Denver, Colorado, USA, 1991.

[3] S. Jajodia, R. Sandhu. Polyinstantiation Integrity in Multilevel Relations. In *Proc. of the IEEE Symposium on Security and Privacy*, Oakland, California, USA, 1988.

[4] K. Smith, M. Winslett. Entity Modeling in the MLS Relational Model. In *Proc. of the 18th VLDB Conference*, Vancouver, BC, Canada, 1992.

[5] R. Sandhu, F. Chen. The Multilevel Relational Data Model. *Transactions on Information and System Security*, Vol. 1, No. 1, 1998.

[6] N. Jukic, S. V. Vrbsky. Asserting Beliefs in MLS Relational Models. *SIGMOD Record*, Vol. 26, No. 3, 1997.

[7] N. Jukic, S. V. Vrbsky, A. Parrish, B. Dixon, B. Jukic. A Belief-Consistent Multilevel Secure Relational Data Model. *Information Systems*, Vol. 24, No. 5, 1999.

[8] Trusted Computer Security Evaluation Criteria, DoD 5200.28-STD. US Department of Defense, 1985.

[9] E. Bell, L. J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report MTR-2997, The Mitre Corporation, Burlington Road, Bedford, MA 01730, USA.

[10] M. D. Abrams, S. Jajodia, H. J. Podell. Information Security An Integrated Collection of Essays. *IEEE Computer Society Press*, Los Alamitos, CA, USA, 1995.

[11] S. Castano, et al. Database Security. *ACM Press*, New York, NY, USA, 1995.

[12] V. Atluri, S. Jajodia, T. F. Keefe, C. MaCollum, R. Mukkamal. Multilevel Secure Transaction Processing: Status and Prospects. *Database Security, X: Status and Prospects*, Chapman & Hall 1997, eds. Pierangela Samarati and Ravi Sandhu.

[13] T. F. Keefe, W. T. Tsai, T. F. Keefe, J. Srivastava. Multilevel Secure Database Concurrency Control. In *Proc. IEEE sixth International Conference on Data Engineering*, Los Angeles, CA, USA, 1990.

[14] L. Lamport. Concurrent Reading and Writing. In *Comm. ACM*, Vol. 20, No. 11, 1997.

[15] P. Ammann, S. Jajodia. A Timestamp Ordering Algorithm for Secure, Single-Version, Multilevel Databases. *Database Security, V: Status and Prospects*, C.E. Landweher, ed., Amsterdam, Holland, 1992.

[16] Oracle Corporation. Trusted Oracle Administrator's Guide. Redwood City, CA, USA, 1992.

[17] Informix. Informix OnLine/Secure Administrator's Guide. Menlo Park, CA, USA, 1993.

[18] E. Bertino, S. Jajodia, L. Mancini, I. Ray. Advanced Transaction Processing in Multilevel Secure File Stores. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, No. 1, 1998.

[19] R. Agrawal, J. Kiernan, R. Srikant, Y. Xu. Hippocratic Databases. In *Proc. of the 28th International Conference on Very Large Databases*, Hong Kong, China, 2002.

[20] Sybase Inc. Building Applications for Secure SQL Server, Sybase Secure SQL Server Release 10.0. Emeryville, CA, USA, 1993.

[21] ISO/IEC 9075:1999. Information-Technology-Database Languages-SQL-Part 1: Framework (SQL/Framework), 1999 .

[22] Cryptek. www.cryptek.com.

INTER-NODE RELATIONSHIP LABELING: A FINE-GRAINED XML ACCESS CONTROL IMPLEMENTATION USING GENERIC SECURITY LABELS

Zheng Zhang
University of Toronto
Toronto, Ontario, Canada
zhzhang@cs.toronto.edu

Walid Rjaibi
IBM Toronto Software Laboratory
Markham, Ontario, Canada
wrjaibi@ca.ibm.com

Keywords: Authorization-transparent, fine-grained access control, label-based access control, XML relationship labeling.

Abstract: Most work on XML access control considers XML nodes as the smallest protection unit. This paper shows the limitation of this approach and introduces an XML access control mechanism that protects inter-node relationships. Our approach provides a finer granularity of access control than the node-based approaches(*i.e.*, more expressive). Moreover, our approach helps achieve the "need-to-know" security principle and the "choice" privacy principle. This paper also shows how our approach can be implemented using a generic label infrastructure and suggests algorithms to create/check a secure set of labeled relationships in an XML document.

1 INTRODUCTION

XML has rapidly emerged as the standard for the representation and interchange of business and other sensitive data on the Web. The current trend of adding XML support to database systems poses new security challenges for an environment in which both relational and XML data coexist. In particular, fine-grained access control is even more necessary for XML than for relational data, given the more flexible and less homogeneous structure of XML data compared to relational tables and rows. The additional difficulty of controlling access over XML data compared to relational data can be summarized as follows.

- The semi-structured nature of XML data, where a schema may be absent, or, even if it is present, may allow much more flexibility and variability in the structure of the document than what is allowed by a relational schema.
- The hierarchical structure of XML, which requires specifying, for example, how access privileges to a certain node propagate from/to the node's ancestors and descendants.

In almost all of the work on XML access control (Bertino and Ferrari, 2002; Damiani et al., 2002; Fan et al., 2004), the smallest unit of protection is the XML node of an XML document, which are specified by XPath fragments. Access to ancestor-descendant and sibling relationships among nodes has not been considered. An access control policy consists of positive (resp. negative) authorization rules that grant (resp. deny) access to some nodes of an XML document. The main difference between XML access control models lies in privilege propagation. Some (Bertino and Ferrari, 2002; Gabillon and Bruno, 2001) forbid access to the complete subtree rooted at an inaccessible node. Alternatively, if a node is granted access while one of its ancestor nodes is inaccessible, the ancestor node would be masked as an empty node in the XML document (Damiani et al., 2002). However, this makes visible the literal of the forbidden ancestor in the path from the root to that authorized node. This can be improved by replacing the ancestor node literal by a dummy value (Fan et al., 2004). However, this still does not solve the problem that different descendant nodes may require their ancestor's literal to be visible or invisible differently. From the differences among the above models, it is clear that defining a view that precisely describes the path leading to an authorized node is difficult. The question that begs to be asked is therefore the following: Is a node the most fine-grained entity within an XML document upon which a fine-grained access control model for XML is to be built?

We believe that the answer to this question is an unequivocal NO. We contend that the path between nodes is a better alternative upon which a fine-grained access control model for XML is to be built (Kanza et al., 2006). In other words, we con-

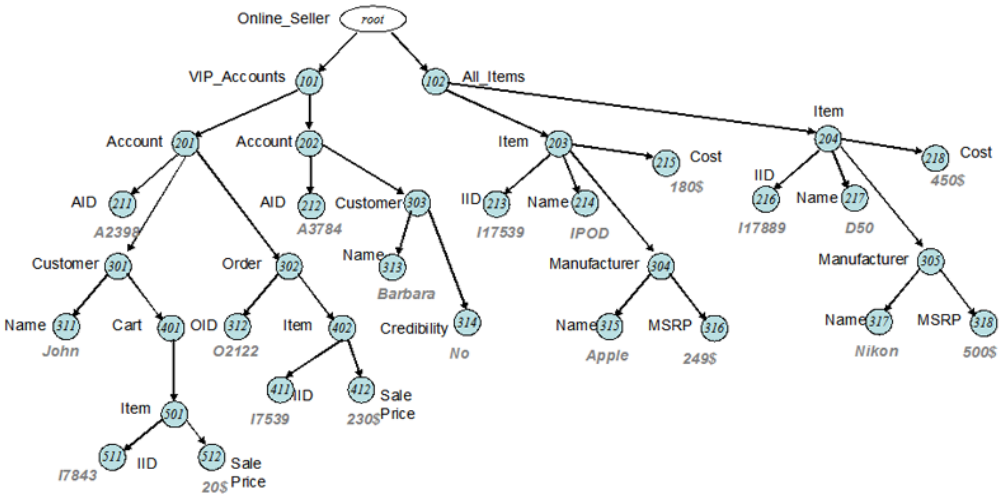


Figure 1: A document that contains information on accounts, orders and items for an online seller.

tend that ancestor-descendent relationships and sibling relationships should be considered as legitimate elements to be protected. The main advantages of our approach are as follows.

First of all, blocking access to a node can be addressed by blocking access to all the relationships relating to the node. For example, in Figure 1, if we want to block all access to the Account Node "202", we could simply block access to all the paths from that node's ancestors to the node and all the paths from the node to its sibling and descendants.

Second, blocking access to relationships helps achieve the "need-to-know" principle, one of the most fundamental security principles. This principle requires information to be accessed by only those who strictly need the information to carry out their assignments. The practice of "need-to-know" limits the damage that can be done by a trusted insider who betrays our trust. The hierarchical structure of an XML document often reveals classification information. For example, in Figure 1, the root of the left subtree of the document represents a special account type "VIP_Accounts". Knowing an account node, say Node "201", belongs to that subtree reveals the account type. If the smallest protection unit is a node, once we let the root of the subtree accessible, we may leak unnecessary information. For example, suppose that the relationship between the Account Node "202" and the account type "VIP_Accounts" at the root of the subtree should be protected, knowing the account type of Node "201" in the subtree reveals the account type of Node "202". With relationship protection, we identify that the ancestor-descendant relationship between Node "101" and Node "202", and the sibling relationship between Node "201" and Node "202" should be protected while we let the ancestor-descendant relationship between Node "101" and Node "201" be accessible.

Third, blocking access to relationships helps achieve the "choice" principle, one of the most fundamental privacy principles. At its simplest, the principle means giving clients options as to how any personal information collected from them may be used. If the smallest protection element is a node, access control over one node is propagated to its ancestor/descendant nodes (Murata et al., 2003), *i.e.*, whenever access is denied to a node, access is denied to its descendants; whenever access is granted to a node, access is granted to all its ancestors. Hence, negative access control policies over ancestor nodes give a common authorized view of the paths leading to their descendants. This violates the "choice" principle: in Figure 1, a client may want to hide the account type but not the other account information for the account with AID "A2398". If the smallest protection element is a relationship between nodes in an XML document, we could protect the relationships between Node "101" and the nodes in the subtree rooted at Node "201", and the sibling relationship between Node "201" and Node "202". Then all the account information except the account type is still accessible from the root of the document tree. Moreover, there is no way to know that the subtree rooted at Node "201" is a subtree of Node "101".

Last but not least, protecting relationships between nodes in an XML document is more expressible in terms of access control policy translation.

Contributions: The contributions made in this paper can be summarized as follows:

1. We propose an authorization-transparent fine-

grained access control model that protects the ancestor-descendant and sibling relationships in an XML document. Our model distinguishes two levels of access to relationships, namely the *existence access* and the *value access*.

2. We propose a new semantics for concealing relationships in an XML document where a relationship is defined by a path in the document.
3. We propose a generic and flexible label-based access control mechanism to protect relationships. Our mechanism allows DBAs to define label-based access control policies.
4. We propose a new query evaluation mechanism to enforce our access control model.
5. We develop algorithms to check/create a secure set of labeled relationships of an XML document.

2 RELATED WORK

XML access control has been studied on issues such as granularity of access, access-control inheritance, default semantics, overriding, and conflict resolutions (Bertino and Ferrari, 2002; Damiani et al., 2002; Gabillon and Bruno, 2001; Murata et al., 2003). In particular, a useful survey of these proposals is given in (Fundulaki and Marx, 2004), which uses XPath to give formal semantics to a number of different models in a uniform way, making it possible to compare and contrast them. Almost all the recent models (Bertino and Ferrari, 2002; Damiani et al., 2002; Gabillon and Bruno, 2001) propose to restrict the user's view of a document by access control policies. In particular, authors in (Damiani et al., 2002; Gabillon and Bruno, 2001) mark each node as "accessible" or "inaccessible" in an XML document and apply conflict resolution policies to compute an authorized pruned view of the document. An alternative approach (Miklau and Suciu, 2003) defines access control policies as XQuery expressions. A user is given a modified document with encrypted data and queries are posed on this modified document. They present a new query semantics that permits a user to see only authorized data. In (Fan et al., 2004), security is specified by extending the document DTD with annotations and publishing a modified DTD. Similarly, work by Bertino *et al*. (Bertino et al., 2001) and Finance *et al*. (Finance et al., 2005) provides XML-based specification languages for publishing secure XML document content, and for specifying role-based access control on XML data (Bhatti et al., 2004; Wang and Osborn, 2004). Restricting access to nodes has also been used in XACL (IBM, 2001) and XACML (Oasis., 2005), two proposed industrial standards. Kanza *et al*. propose to restrict access to ancestor-descendant relationships (Kanza et al., 2006) and introduce authorization-transparent access control for XML data under the Non-Truman model (Rizvi et al., 2004).

3 DATA MODEL AND QUERIES

We consider an XML document as a rooted directed tree over a finite set of node literals L with a finite set of values A attached to atomic nodes (*i.e*., nodes with no outgoing edges). Formally, a document D is a 5-tuple $(N_D, E_D, root_D, literal\_of_D, value\_of_D)$, where N_D is a set of nodes, E_D is a set of directed edges, $root_D$ is the root of a directed tree, $literal\_of_D$ is a function that maps each node of N_D to a literal of L, and $value\_of_D$ is a function that maps each atomic node to a value of A. In order to simplify the data model, we do not distinguish between elements and attributes of an XML document. We also assume that all the values on atomic nodes are of type *PC-DATA* (*i.e*., *String*).

Example 3.1 Figure 1 shows a document that contains information on accounts, orders, and items for an online seller. Nodes are represented by circles with ID's for easy reference. Values in A appear below the atomic nodes and are written in bold font.

In this paper, we use XPath (Clark and DeRose, 1999) for formulating queries and specifying relationships. XPath is a simple language for navigation in an XML document. In XPath, there are thirteen types of axes that are used for navigation. Our focus is on the *child* axis (/), the *descendant-or-self* axis (//), the *preceding-sibling* axis and the *following-sibling* axis that are the most commonly used axes in XPath. Our model, however, can also be applied to queries that include the other axes.

4 RELATIONSHIP ACCESS

First, we consider what it means to conceal a relationship. In general, a *relationship* is an undirected path between two nodes in an XML document. A set of *relationships* is represented by two sets of nodes. For example, the pair (C, N), where C is the set of all Customer Nodes and N is the set of all Name Nodes in Figure 1, represents the set of relationships between customers and their names. Concealing the relationships (C, N) means that for every customer c and name n in the document, a user will not be able to infer (with certainty), from any query answers, whether n is the name for c. We want this to be true for all authorized query results. Note that we are concealing the presence or absence of relationships, so we are

concealing whether any of the set of pairs in (C, N) exists in the document.

Definition 4.1 (Concealing a Relationship) *Given an XML document D and a pair of nodes n_1 and n_2 in D, the relationship (n_1, n_2) is concealed if there exists a document D' over the node set of D, such that the following is true.*

1. *Exactly one of D and D' has a path from n_1 to n_2.*
2. *For any XPath query Q, the authorized answer set of Q over D is equal to that of Q over D'.*

We consider two kinds of relationships in an XML document, namely the ancestor-descendant relationships and the sibling relationships. Kanza *et al*. consider ancestor-descendant relationships only (Kanza et al., 2006). Sibling relationships are inferred by the ancestor-descendant relationships. Hence, when access to an ancestor-descendant relationship is blocked in their model, access to the related sibling relationships is automatically blocked.

Example 4.2 In Figure 1, suppose the relationship between VIP_Accounts Node "101" and Account Node "201" is inaccessible, then the sibling relationship between Node "201" and Node "202" is lost.

It could be necessary to preserve such sibling relationship information. For example, one policy may want to block access to the ancestor-descendant relationships between VIP_Accounts Node and Account Nodes while maintain access to the sibling relationships between the Account Nodes.

On the other hand, it might be desirable to block access to sibling relationships only. For example, one policy may want to block access to the sibling relationship between Customer and his Order.

In order to express such access control policies, we consider sibling relationships as well as ancestor-descendant relationships.

We distinguish two levels of access to relationships, namely the *existence access* and the *value access*. In value access, information about a relationship indicates a node whose ID is "v_a" and whose literal is "A" is related to a node whose ID is "v_b" and whose literal is "B". For example, the pair (C, N) is a value access to the relationships between Customer Nodes and Name Nodes. In existence access, information about a relationship is basically the same as information of value access but lacks at least one of the values "v_a" and "v_b". In other words, existence access to a relationship returns whether a node of some literal is related to some node. For example, existence access could indicate a node whose literal is "A" is related to a node whose literal is "B". Obviously, if a relationship is not accessible under existence access, then the relationship is not accessible under value access.

Example 4.3 Consider the relationship between the account with AID "A2398" and its customer name in Figure 1. The value access to this relationship returns that Node "201" whose literal is "Account" is related to Node "311" whose literal is "Name" and whose value is "John". The typical queries that will return this information are:

Q_1: *//Account[AID="A2398"]*,
Q_2: *//Account[AID="A2398"]/Customer/Name*.

Now consider an existence access to this relationship: a query Q_3 wants to return all the accounts' AID's that have a customer name. The fact that "A2398" is returned tells us that there exists a customer with name under the account with AID "A2398", but it does not tell us what the customer's name is, nor the Node ID "311". In other words, Q_1 and Q_3 reveal that Node "201" whose literal is "Account" is related to some node n whose literal is "Name", where n is a child of some node whose literal is "Customer" and which is a child of Node "201".

Q_3: *//Account[Customer/Name]/AID*.

In the next section, we show how to specify ancestor-descendant and sibling relationships and attach access labels to them.

5 ACCESS CONTROL POLICY SPECIFICATION

Our access control model uses a generic, flexible label infrastructure (Rjaibi and Bird, 2004) where a label has only one component "access level". The value of the component can be "EXISTENCE", "VALUE", or "NULL". The ranks of these values are as follows: "EXISTENCE" > "VALUE" > "NULL". We distinguish two types of labels: *Access labels* and *Path labels*. Access labels are created and assigned to database users, roles, or groups along with the type of access for which the access label is granted (*i.e*., Read/Write). For simplicity, we consider only users in this paper. We call *read (resp. write) access label* an access label associated with the Read (resp. Write) access type. Path labels are created and attached to paths of an XML document. When a user or a path is not associated with a label, the "NULL" label is assumed for that user or path.

Example 5.1 The following statement creates and grants the "EXISTENCE" access label to a database user Mike for the Read access type.

GRANT ACCESS LABEL EXISTENCE
TO USER Mike FOR READ ACCESS

The following statement revokes the "EXISTENCE" read access label from Mike.

REVOKE ACCESS LABEL EXISTENCE
FROM USER Mike FOR READ ACCESS

Access to an XML document is based upon the labels associated with the paths of the XML document and the label associated with the user accessing the document via the paths. A label access policy consists of label access rules that the database system evaluates to determine whether a database user is allowed access to an XML document. Access rules can be categorized as Read Access rules and Write Access rules. The former is applied by the database system when a user attempts to read a path in an XML document; the latter is applied when a user attempts to insert, update or delete a path in an XML document. In both cases, a label access rule is as follows:

Access Label ⟨operator⟩ Path Label

where the operator is one of the arithmetic comparison operators $\{=, \leq, <, >, \geq, \neq\}$.

Example 5.2 The following statement creates a label access policy that *(1)* does not allow a user to read a path unless his read access label is larger than or equal to the path label, *(2)* does not allow a user to write a path unless his write access label is equal to the path label.

CREATE LABEL POLICY XML-FGAC
READ ACCESS RULE rule
READ ACCESS LABEL $\geq$ Path LABEL
WRITE ACCESS RULE rule
WRITE ACCESS LABEL $=$ Path LABEL

Recall value access to a relationship returns more information than existence access. An "EXISTENCE" label protects existence and value access. A "VALUE" label protects value access only. Therefore, if a user with a "NULL" read access label wants to existence access a path with a "VALUE" path label, access should be allowed since this existence access does not return the complete relationship information from value access. We call this the DEFAULT policy. This policy only applies to Read Access since any Write Access involves real node ID's (*i.e.*, existence access is impossible). This policy could coexist with other policies such as XML-FGAC to give a more complete authorized answer set of a query.

Example 5.3 Assume the relationship in Example 4.3 has a "VALUE" path label. If a user with a "NULL" read access label asks query Q_3, the existence access to the relationship should be allowed.

Next, we introduce how the labels are attached to paths in an XML document. First, attaching a label to ancestor-descendant paths are specified by an SQL statement in the following form:

ATTACH *path_label* ANCS $path_1$ DESC $path_2$,

where $path_1$ and $path_2$ are two XPath expressions. Notice expression $path_2$ is a relative XPath expression w.r.t. $path_1$. The two expressions specify pairs of ancestor nodes (*i.e.*, $path_1$) and descendent nodes (*i.e.*, $path_1/path_2$). Expression *path_label* is a label.

Example 5.4 The following expression attaches "EXISTENCE" path labels to the relationships between Account Nodes and their Customers' Name Nodes in Figure 1.

ATTACH EXISTENCE ANCS *//Account*
DESC */Customer/Name*

The following expression attaches a "VALUE" path label to the relationship between the Item Node with Name "IPOD" and its Cost Node in Figure 1.

ATTACH VALUE ANCS *//Item[Name = "IPOD"]*
DESC *//Cost*

For sibling relationships, we consider the *preceding-sibling axis* and *the following-sibling axis* in XPath. Thus, attaching a label to sibling paths are specified by XPath expressions in the following form:

ATTACH *path_label*
NODE $path_1$ PRECEDING-SIBLING $path_2$
FOLLOWING-SIBLING $path_3$,

where $path_1$, $path_2$ and $path_3$ are three XPath expressions. Notice expressions $path_2$ and $path_3$ are two relative XPath expressions w.r.t. $path_1$. The expressions specify relationships between some nodes (*i.e.*, $path_1$), and their preceding siblings (*i.e.*, $path_1/preceding\text{-}sibling :: path_2$) as well as the relationships between the nodes and their following siblings (*i.e.*, $path_1/following\text{-}sibling :: path_3$). Notice the *PRECEDING-SIBLING* expression and the *FOLLOWING-SIBLING* expression do not have to appear at the same time.

Example 5.5 The following expression attaches a "VALUE" path label to the relationship between the Account whose Customer has Name "Barbara" and its preceding sibling.

ATTACH VALUE
NODE *//Account[Customer/Name = "Barbara"]*
PRECEDING-SIBLING *Account*

Note that the SQL statement to detach a label from an ancestor-descendant path or a sibling path is similar to the SQL statement to attach a label to those paths except that ATTACH is replaced by DETACH.

6 QUERY EVALUATION

In authorization-transparent access control, users formulate their queries against the original database rather than against authorization views that transform and hide data (Motro, 1989). In (Rizvi et al., 2004), authorization transparent access control is categorized into two basic classes, the *Truman model* and the *Non-Truman model*. In the Truman model, an access control language (often a view language) is used to specify what data is accessible to a user. User queries are modified by the system so that the answer includes

only accessible data. Let Q be a user query, D be a database and D_u be the part of D that the user is permitted to see, then query Q is modified to a safe query Q_s such that $Q_s(D) = Q(D_u)$. We call $Q_s(D)$ the *authorized answer set* of Q over D. In contrast, in the Non-Truman model, a query that violates access control specifications is rejected, rather than modified. Only *valid* queries are answered.

Our model is an authorization-transparent Truman model. We allow users to pose XPath queries against the original labeled XML document. The evaluation of an XPath query over a labeled XML document has two parts. First, we change the usual XPath query semantics as follows. If a child axis occurs, the evaluation follows a parent-child path; if a descendant-or-self axis occurs, the evaluation follows an ancestor-descendant path; if a preceding-sibling axis occurs, the evaluation follows a preceding-sibling path; if a following-sibling axis occurs, the evaluation follows a following-sibling path.

Second, we need to make sure that for each path accessed, a user is allowed access to that path based on the path label and the user's access label. Suppose a path P has a path label L_1 and a user Mike has a read access label L_2. According to the XML-FGAC policy, *(1)* if L_2 is "EXISTENCE", Mike could read the path P regardless of the value of label L_1; *(2)* if L_2 is "VALUE", Mike could read the path P if L_1 is not "EXISTENCE"; *(3)* if L_2 is "NULL", Mike can only access paths with "NULL" labels; if the DEFAULT policy coexists, Mike could ask queries to existence access the path P if L_1 is "VALUE". The discussion for Write Access is similar. The above logic is inserted into the query access plan. When the access plan is executed, the access rules from the label access policy associated with the labeled XML document are evaluated for each path accessed in the document. This approach allows the cached access plan to be reused because the access labels of the user who issued the query are acquired during runtime.

For an XML document, there is an ordering, *document order* (Clark and DeRose, 1999), defined on all the nodes in the document corresponding to the order in which the first character of the XML representation of each node occurs in the XML representation of the document. This ordering information may leak information as shown in the following example.

Example 6.1 Let us look at Figure 1 again. Suppose one security policy wants to block public access to the sibling relationships between the Customer Nodes and their Order Nodes. Suppose the following queries are allowed to return their answers in document order: //*Customer* and //*Order*. Then the order of Customer output might match the order of Order output, hence leaks secret information. The situation becomes worse if the document has a registered schema and the schema shows publicly that each customer has a fixed number, say 2, of orders. In this case, the association between a Customer and his Orders is completely leaked.

To prevent an information leak based on document order, we shuffle the output as follows. Each node in the output will receive a random number. And the nodes will be output based on the order of their assigned random numbers.

In sum, the processing algorithm to be inserted in the access plan for a labeled XML document with XML-FGAC and DEFAULT policies is as follows.

Algorithm: Insert Read and Write Access logic into a query access plan for a labeled XML document.

1. Fetch the user's Access Labels for Read and Write actions (*e.g.*, from a system catalog table).
2. For all paths accessed, do the following.
   (a) If it is a Read Access and READ Access rules do not permit access, skip the path unless *(1)* the Read Access Label is "NULL", *(2)* the Path Label is "VALUE", and *(3)* it is an existence access.
   (b) If it is a Write Access and Write Access rules do not permit access, skip the path.
3. Shuffle output.

Example 6.2 Suppose the document in Figure 1 has two labels attached to its paths as specified in Example 5.4 and the label access policies are XML-FGAC and DEFAULT. Suppose a database user Mike with a read access label "EXISTENCE" asks the query Q_1: //*Account[Customer/Name]*. The query access plan checks the following paths:

1. the paths P_1 from the root of the document to Account Nodes, *i.e.*, //*Account*,
2. the paths P_2 from Account Nodes to their descendant Name Nodes via Customer Nodes, *i.e.*, ANCS //*Account* DESC /*Customer*/*Name*,
3. the paths P_3 from Customer Nodes to their children Name Nodes, *i.e.*, *Customer*/*Name*.

Paths P_1 and P_3 have "NULL" labels, hence, access is allowed. Paths P_2 have "EXISTENCE" labels. Mike could read them since his read access label is "EXISTENCE". Read access to P_2 is denied for any other labels and the authorized answer set is empty.

Next, suppose another user John with a read access label "VALUE" asks the query Q_2: //*Item*//*Cost*. The query access plan checks the following paths:

1. the paths P_1 from the root of the document to the Item Nodes, *i.e.*, //*Item*,
2. the paths P_2 from the Item Nodes to their descendant Cost Nodes, *i.e.*, ANCS //*Item* DESC //*Cost*.

Paths P_1 have "NULL" labels, hence, access is allowed. For P_2, one path P_{21} has a "NULL" label; the other path P_{22} has a "VALUE" label as it is

ANCS *//Item[Name="IPOD"]* DESC *//Cost*. John could read P_2 if his read access label is "VALUE". John could read P_{21} but not P_{22} if his read access label is "NULL". Hence, the authorized answer set is "450$". However, even if John's read access label is "NULL", the following query from John will still return the complete answer to Q_3: *//Item[Cost]*. This is because Q_3 only existence accesses the paths P_2, *i.e.*, the authorized answer set only indicates there exist Cost children Nodes for the Item Nodes "203" and "204", but no information about the values and node ID's of the Cost Nodes is leaked.

7 CREATE A SECURE SET OF LABELED RELATIONSHIPS

Our goal is to allow users to label node relationships and let them be sure that what they want to conceal is truly concealed from the users whose access labels do not satisfy the label access policy with the path labels. Unfortunately, it is impossible to guarantee concealment for any arbitrary set of relationships. Sometimes, it is possible to infer a concealed relationship from the relationships that are not concealed.

Let us see an example of four cases where a relationship could be inferred from a pair of non-concealed relationship.

Example 7.1 In Figure 1, suppose it is known that Account Node "201" is a descendant of VIP_Accounts Node "101" and Customer Node "301" is a descendant of Account Node "201". Then, there is no point to conceal the ancestor-descendant relationship between VIP_Accounts Node "101" and Customer Node "301".

Suppose it is known that Customer Node "301" is a descendant of VIP_Accounts Node "101" as well as Account Node "201". Since there is only one path from the root of the document to Account Node "201", there is no point to conceal the ancestor-descendant relationship between VIP_Accounts Node "101" and Account Node "201".

Suppose it is known that Account Node "201" and Account Node "202" are the children of VIP_Accounts Node "101", then there is no point to conceal the sibling relationship between Account Node "201" and Account Node "202".

Suppose it is known that VIP_Accounts Node "101" has a descendant Customer Node "301" and the customer has a sibling Order Node "302", then there is no point to conceal the ancestor-descendant relationship between VIP_Accounts Node "101" and Order Node "302".

We say a set of labeled relationships/paths in an XML document D is *not secure* w.r.t. a path label L if one of the following four cases happens.

1. Case 1: D has three nodes, n_1, n_2 and n_3 s.t. the ancestor-descendant path from n_1 to n_2 and the ancestor-descendant path from n_2 to n_3 have labels $L_{12} < L$ and $L_{23} < L$. The ancestor-descendant path from n_1 to n_3 has a label $L_{13} \geq L$.
2. Case 2: D has three nodes, n_1, n_2 and n_3 s.t. the ancestor-descendant path from n_1 to n_3 and the ancestor-descendant path from n_2 to n_3 have labels $L_{13} < L$ and $L_{23} < L$. The ancestor-descendant path from n_1 to n_2 has a label $L_{12} \geq L$.
3. Case 3: D has three nodes, n_1, n_2 and n_3 s.t. n_1 is the parent of n_2 and n_3, the parent-child path from n_1 to n_2 and the parent-child path from n_1 to n_3 have labels $L_{12} < L$ and $L_{13} < L$. The sibling path from n_2 to n_3 has a label $L_{23} \geq L$ or the sibling path from n_3 to n_2 has a label $L_{32} \geq L$.
4. Case 4: D has three nodes, n_1, n_2 and n_3 s.t. the ancestor-descendant path from n_1 to n_2 has a label $L_{12} < L$, and either the sibling path from n_2 to n_3 has a label $L_{23} < L$ or the sibling path from n_3 to n_2 has a label $L_{32} < L$. The ancestor-descendant path from n_1 to n_3 has a label $L_{13} \geq L$.

There is a simple test to verify that a set of labeled relationships/paths in an XML document D is not secure w.r.t. a path label L. The test starts by computing three ternary relations R_1, R_2 and R_3. The first two columns store the start/end nodes of paths. The third column stores the label associated with paths (if a label is missing, then it is a NULL value). In particular, R_1 stores all ancestor-descendant paths in D, R_2 stores all parent-child paths in D, and R_3 stores all sibling paths in D.

1. Case 1 is true for a path label L iff the expression $\pi_{\$1,\$5}(R_{1,L} \bowtie_{\$2=\$1} R_{1,L}) - R_{1,L}$ is not empty where $R_{1,L}$ is $\sigma_{\$3<L}(R_1)$.
2. Case 2 is true for a path label L iff the expression $\pi_{\$1,\$4}(R_{1,L} \bowtie_{\$2=\$2} R_{1,L}) - R_{1,L}$ is not empty where $R_{1,L}$ is $\sigma_{\$3<L}(R_1)$.
3. Case 3 is true for a path label L iff the expression $\pi_{\$2,\$5}(R_{2,L} \bowtie_{\$1=\$1} R_{2,L}) - R_{3,L}$ is not empty where $R_{2,L}$ is $\sigma_{\$3<L}(R_2)$ and $R_{3,L}$ is $\sigma_{\$3<L}(R_3)$.
4. Case 4 is true for a path label L iff the expression $\pi_{\$1,\$5}(R_{1,L} \bowtie_{\$2=\$1} R_{3,L}) - R_{1,L}$ is not empty where $R_{1,L}$ is $\sigma_{\$3<L}(R_1)$ and $R_{3,L}$ is $\sigma_{\$3<L}(R_3)$.

Furthermore, we give intuitive conditions to construct a secure set of labeled relationships for an XML document. If we ignore the directions of ancestor-descendant and sibling paths, all these paths form cycles in an XML document. To assign a path label L to a relationship between two nodes n_1 and n_2 in an XML document D, we must make sure, for every cycle that includes the path from n_1 to n_2, either there is another path whose label $L' \geq L$, or n_1 and n_2 are descendants of some nodes in the cycle and n_1, n_2 are

not children of the same parent. Both cases ensure there is uncertainty whether a relationship between two nodes n_1 and n_2 exists: the first case by having another path missing in the cycle, while in the second case, the fact that n_1 and n_2 are descendants of some nodes in the cycle introduces uncertainty except when they are children of the same parent, in which case the sibling relationship between n_1 and n_2 is leaked.

There is another possible information leak due to *singleton-source disclosure* (Kanza et al., 2006). In short, a user can infer that two nodes n_1 and n_2 are related in a document D when *(1)* the path from the root of document D to node n_2 must go through a node whose literal is A, *(2)* the only node with literal A in document D is node n_1. An algorithm to test singleton-source disclosure has been proposed in (Kanza et al., 2006) and we will not repeat it here.

8 CONCLUSION

This paper has introduced a fine-grained access control model for XML data using generic security labels. Our model is based on inter-node relationship labeling and provides finer-grained access control than traditional node labeling approaches, hence helps achieve the "need-to-know" security principle and the "choice" privacy principle. We propose a new semantics for concealing relationships in an XML document under the Truman model. To enforce our model, we provide a new query evaluation algorithm and suggest algorithms to check/create a set of secure labeled paths for an XML document.

Our future work includes implementing our model and validating its effectiveness and performance using real-life XML access control user cases. An important challenge is adapting our mechanism to XQuery, general XML document graphs and XML schemas.

Acknowledgements: We thank NSERC and IBM Toronto CAS for their support, and Renée J. Miller for her careful comments.

Trademark: IBM is a trademark or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

Disclaimer: The views expressed in this paper are those of the authors and not necessarily of IBM Canada Ltd. or IBM Corporation.

REFERENCES

Bertino, E., Castano, S., and Ferrari, E. (2001). On specifying security policies for web documents with an xml-based language. In *SACMAT*, pages 57–65.

Bertino, E. and Ferrari, E. (2002). Secure and selective dissemination of xml documents. *ACM Trans. Inf. Syst. Secur.*, 5(3):290–331.

Bhatti, R., Bertino, E., Ghafoor, A., and Joshi, J. (2004). Xml-based specification for web services document security. In *IEEE Computer*, volume 4 of *37*, pages 41–49.

Clark, J. and DeRose, S. (1999). XML Path Language (XPath) version 1.0. Available at `http://www.w3.org/TR/xpath`.

Damiani, E., de C. di Vimercati, S., Paraboschi, S., and Samarati, P. (2002). A fine-grained access control system for xml documents. *ACM Trans. Inf. Syst. Secur.*, 5(2):169–202.

Fan, W. F., Chan, C. Y., and Garofalakis, M. N. (2004). Secure xml querying with security views. In *SIGMOD*, pages 587–598.

Finance, B., Medjdoub, S., and Pucheral, P. (2005). The case for access control on xml relationships. Technical report, INRIA. Available from http://www-smis.inria.fr/dataFiles/FMP05a.pdf.

Fundulaki, I. and Marx, M. (2004). Specifying access control policies for xml documents with xpath. In *SACMAT*, pages 61–69.

Gabillon, A. and Bruno, E. (2001). Regulating access to xml documents. In *Working Conference on Database and Application Security*, pages 311–328.

IBM (2001). Xml access control. *http://xml.coverpages.org/xacl.html*.

Kanza, Y., Mendelzon, A., Miller, R., and Zhang, Z. (2006). Authorization-transparent access control for xml under the non-truman model. In *EDBT*, pages 222–239.

Miklau, G. and Suciu, D. (2003). Controlling access to published data using cryptography. In *VLDB*, pages 898–909.

Motro, A. (1989). An access authorization model for relational databases based on algebraic manipulation of view definitions. In *ICDE*, pages 339–347.

Murata, M., Tozawa, A., Kudo, M., and Hada, S. (2003). Xml access control using static analysis. In *CCS*, pages 73–84. ACM Press.

Oasis. (2005). Oasis exensible access control markup language (xacml 2.0). *http://www.oasis-open.org/committees/xacml*.

Rizvi, S., Mendelzon, A., Sudarshan, S., and Roy, P. (2004). Extending query rewriting techniques for fine-grained access control. In *SIGMOD*, pages 551–562.

Rjaibi, W. and Bird, P. (2004). A multi-purpose implementation of mandatory access control in relational database management systems. In *VLDB*, pages 1010–1020.

Wang, J. Z. and Osborn, S. L. (2004). A role-based approach to access control for xml databases. In *SACMAT*, pages 70–77.

An Introduction to Multilevel Secure Relational Database Management Systems

Walid Rjaibi

IBM Toronto Software Laboratory
Markham, Ontario, Canada
wrjaibi@ca.ibm.com

Abstract

Multilevel Security (MLS) is a capability that allows information with different classifications to be available in an information system, with users having different security clearances and authorizations, while preventing users from accessing information for which they are not cleared or authorized. It is a security policy that has grown out of research and development efforts funded mostly by the U.S. Department of Defense (DoD) to address some of the drawbacks of the single level mode of operation that was used at the DoD. The goal was to build and deploy an MLS-compliant environment (*e.g.,* Networks, Operating Systems, Database Systems) that would provide a much needed efficiency in processing and distributing classified information by providing security through computer security, communications security, and trusted system techniques instead of using physical controls, administrative procedures, and personnel security. As Relational Database Management Systems (RDBMS) are at the heart of the DoD's information system, significant research and development efforts have been put into building multilevel secure RDBMS, which have led to the emergence of a number of multilevel secure RDBMS solutions, including commercial ones. Over the past few years and with the increase of security concerns, MLS compliance has become a major requirement from a number U.S. Federal Government agencies that appear to have grown beyond the traditional agencies that require such type and level of security. This paper introduces MLS, and outlines the challenges and complexities of building a multilevel secure RDBMS. The paper also gives concrete examples of both research and commercial multilevel secure RDBMS and describes how they met the above challenges and complexities.

1 Introduction

Multilevel Security (MLS) is a capability that allows information with different classifications to be available in an information system, with users having different security clearances and authorizations, while preventing users from accessing information for which they are not cleared or authorized[2]. It is a security policy that has been developed primarily for the U.S. military and intelligence communities, but has also been adopted by some civilian organizations that store, process and distribute classified information (*e.g.,* major aircraft manufacturers) as well as by a number of defense departments around the world.

Given the extremely high value of the information that could be stored in a military or intelligence database, and the potential damage that could result from the unauthorized disclosure, alteration or loss of such information, preventing users from accessing information for which they are not cleared or authorized requires much more than just implementing an access control policy. In particular, security guards must be put in place to prevent users from gaining access to information for which they are not cleared or authorized through indirect means.

Covert channels[5] are examples of such indirect means. A covert channel can easily be established with conventional database concurrency control algorithms such as two-phase locking (2PL) and timestamp ordering (TO)[6]. In both 2PL and TO algorithms, whenever there is contention for the same data item by transactions executing at different security levels, a lower level transaction may be either delayed or suspended to ensure correct execution. In such a scenario, two colluding transactions executing at high and low security levels can establish an information flow channel from a high security level to a low security level by accessing selected data items according to some agreed-upon code[4].

Inference[7] is another indirect means by which users can gain knowledge about information for which they are not cleared or authorized. For example, enforcing a primary key constraint[6] across data from different security levels could allow a non sufficiently cleared user to gain knowledge about the existence of a data row at a higher security level from the duplicate key error message that is returned to that user when he or she attempts to insert a data row at a low security level but having the same primary key as the data row at the higher security level.

Building a multilevel secure RDBMS has thus posed significant challenges to the database research community. For instance, secure database transaction protocols had to be developed, and a solution to reconcile the conflicting requirements between data integrity and confidentiality had to be found. MLS has also posed significant challenges to database vendors as building a multilevel secure RDBMS often requires rebuilding major portions of an existing commercial RDBMS.

There has been an abundance of research within the last two decades or so in the area of multilevel secure RDBMS. Such research has addressed specific aspects of building a multilevel secure RDBMS such as secure transaction protocols, system architectures, or *polyinstantiation*[8], and there is a rich set of publications about those specific aspects[8, 4, 9, 10]. However, the multilevel secure RDBMS research literature surprisingly lacks the kind of publication that would allow someone to get a good understanding about what it takes to build a multilevel secure RDBMS as a whole, as well as to serve as a quick guide for those who might be thinking about building such RDBMS.

Moreover, the term multilevel security is heavily overloaded across the Information Technology (IT) industry and often means different things to people from different backgrounds as there are not only multilevel secure RDBMS, but also multilevel secure operating systems, multilevel secure networks, multilevel secure webservers, etc. In addition to being heavily overloaded, MLS is often incorrectly used interchangeably with emerging marketing terms such as Label-Based Access Control (LBAC), Row-Level Security, and others. All of this makes it extremely difficult for those who have not been directly involved in designing or building a multilevel secure RDBMS to get a good understanding about what it really takes to build a multilevel secure RDBMS.

In this paper, the author wishes to share his expertize in database security and privacy to try to clarify the mystery of multilevel security, as well as to outline the challenges and complexities of building a multilevel secure RDBMS.

1.1 Synopsis

The rest of this paper is organized as follows. Section 2 introduces MLS and describes the MLS certification and evaluation process. Section 3 presents and compares Multilevel Secure RDBMS architectures. Section 4 describes the issue of polyinstantiation. Section 5 presents multilevel secure transaction pro-

cessing. Section 6 gives concrete examples of both research and commercial multilevel secure RDBMS. Lastly, section 7 summarizes the concepts introduced in this paper.

2 What is Multilevel Security?

A good understanding of MLS would not be complete without understanding its origins, and what problems it was meant to solve. The U.S. military and intelligence communities have historically segregated data based upon its security classification. Classified data must reside and be processed on dedicated systems that do not provide access to users outside of the immediate community of interest and are often separated by an *air gap* and connected only by a *sneaker net*[2]. The main drawbacks of such operational scheme can be summarized as follows:

- Redundant databases: To store data with different security levels (*e.g.,* Top Secret data and Unclassified data), a separate database must be created and maintained for each security level.
- Redundant workstations: A user who is required to access data with different security levels (*e.g.,* Top Secret data and Unclassified data) would be required to use a different workstation to access each type of data.
- High cost of IT infrastructure: It is not possible to share the computer and communication system infrastructures, such as cabling, network components, printers, and workstations without risking to compromise security.
- Inefficiency: Staff members need to access several systems to perform their duties.

The U.S. DoD has therefore funded significant research and development projects across various organizations to come up with a solution that would allow classified information to be stored, processed and distributed in a secure way, but without the drawbacks listed above. MLS was that solution[2]. MLS allows information with different classifications to be available in an information system, with users having different security clearances and authorizations, while preventing users from accessing information for which they are not cleared or authorized[2]. For example, an MLS system might process both Secret and Top Secret collateral data and have some users whose maximum clearance is Secret and others whose maximum clearance is Top Secret. Another MLS system might have all its users cleared at the Top Secret level, but have the ability to release information classified as Secret to a network consisting of only Secret users and systems. In each of these instances, the system must implement mechanisms to provide assurance that the system's security policy is strictly enforced. MLS has resulted in a shift from providing security through physical controls, administrative procedures, and personnel security to providing security using computer and communication security.

2.1 The Bell-LaPadula Multilevel Security Model

The Basic model of MLS was first introduced by Bell and LaPadula[11]. The model is stated in terms of objects and subjects. An object is a passive entity such as a data file, a record, or a field within a record. A subject is an active process that can request access to objects. Every object is assigned a classification, and every subject a clearance. Classifications and clearances are collectively referred to as labels. A label is a piece of information that consists of two components: A hierarchical component and a set of unordered compartments. The hierarchical component specifies the sensitivity of the data. For example, a military organization might define levels Top Secret, Secret, Confidential and Unclassified. The compartments component is nonhierarchical. Compartments are used to identify areas that describe the sensitivity or category of the labeled data. For example, a military organization might define compartments NATO, Nuclear and Army. Labels are partially ordered in a lattice as follows: Given two labels L_1 and L_2, $L_1 >= L_2$ if and only if the hierarchical component of L_1 is greater than or equal to that of L_2, and

the compartment component of L1 includes the compartment component of L_2. L_1 is said to *dominate* L_2. MLS imposes the following two restrictions on all data accesses:

- The Simple Security Property or "No Read Up": A subject is allowed a read access to an object if and only if the subject's label dominates the object's label.
- The *-Property (pronounced the star property) or "No Write Down": A subject is allowed a write access to an object if and only if the object's label dominates the subject's label.

2.2 Evaluation and Certification

Multilevel secure systems must complete an extensive evaluation and certification process before they can be used in military applications. The evaluation and certification of a multilevel secure system is usually conducted by an independent testing laboratory and is based upon a clearly defined set of criterion. One set of criteria is called common criteria, which has recently been adopted as an ISO standard[3]. Another set of evaluation criteria used by the U.S. DoD is the Trusted Computer System Evaluation Criteria (TCSEC)[5]. Most multilevel secure RDBMS have been developed before common criteria was adopted. TCSEC has been the norm for evaluating such RDBMS.

TCSEC is divided into four divisions: D, C, B, and A ordered in a hierarchical manner with the highest division (A) reserved for systems providing the most comprehensive security. Each division represents a major increase in the overall confidence, or trust, that one can place in the system. Successive levels of trust build upon and incorporate the criteria of the previous lower level of trust.

Within Divisions C and B there are a number of subdivisions known as classes. The classes are also ordered in a hierarchical manner with systems representative of Divisions C and B characterized by the set of computer security mechanisms that they possess. For Division C, Discretionary Access Control (DAC)[6] is provided, whereby users can grant or deny access by other users and groups of users to the system resources that the users control. For Division B, Mandatory Access Control (MAC)[1] is provided. MAC employs the simple security property and the *-property of the Bell-LaPadula MLS model to protect data of different security levels. Division A also provides the MAC features.

Systems representative of the higher classes in Division B and Division A derive their security attributes more from their design and implementation structure than merely security features or functionality. Increased assurance that the required features are operative, correct, and tamperproof under all circumstances is gained through progressively more rigorous design, implementation, and analysis during the development process. Division A requires formal (e.g., mathematical) design and verification techniques to provide increased assurances over Division B.

Multilevel secure systems are associated with TCSEC divisions B and A[2].

3 Multilevel Secure RDBMS Architectures

Multilevel secure RDBMS architectures can be divided into two general types, depending on whether mandatory access control is enforced by the RDBMS itself or delegated to a trusted operating system. These two general types are the Woods Hole Architecture and the Trusted Subjects Architecture[9, 10].

3.1 Woods Hole Architectures

The Woods Hole architectures are the outcome of a three-week study on trusted data management sponsored by the U.S. Air Force at Woods Hole, Massachusetts, USA in 1982[9, 10]. The subject of this study was the following: Can we build a multilevel secure RDBMS using existing untrusted off-the-shelf RDBMS, with minimal change?

The Woods Hole architectures assume that an untrusted off-the-shelf RDBMS is used to access data and that trusted code is developed around that RDBMS to provide an overall secure RDBMS. They can be divided into two main categories: The kernelized architectures and the distributed architectures[9, 10].

3.1.1 Kernelized Architectures

The kernelized architecture[9, 10] uses a trusted operating system and multiple copies of an off-the-shelf RDBMS, where each copy is associated with some trusted front-end. Each pair (trusted front-end, RDBMS) is associated with a particular security level. The trusted operating system enforces its full access control policy on all accesses by the RDBMS to the RDBMS objects. It ensures that data at different security levels is stored separately, and that each copy of the RDBMS gets access to data that is authorized for its associated security level. The latter is possible because the multilevel database is decomposed into multiple single-level databases, where each represents a fragment of the conceptual multilevel database. Each fragment is stored in a single-level operating system object (*e.g.,* a file) which is labeled by the operating system at the corresponding security level, and thus can only be accessed according to the MAC policy of the operating system.

Figure 1 illustrates a kernelized architecture where one RDBMS is associated with the security level "High" and another RDBMS is associated with the security level "Low". The RDBMS associated with the security level "High" has access to both the fragment of the database at the high security level and the fragment of the database at the low security level. But the RDBMS associated with the security level "Low" has access only to the fragment of the database at the low security level.

A benefit of this architecture is that data at different security levels is isolated in the database, which allows for higher level assurance. Another benefit is that, assuming an already evaluated operating system, this architecture should minimize the amount of time and effort to evaluate the RDBMS. However, this architecture results in an additional overhead as the trusted operating system needs to separate data at different security levels when it is added to the database and might also need to combine data from different security levels when data is retrieved by an RDBMS copy that is associated with a high security level.

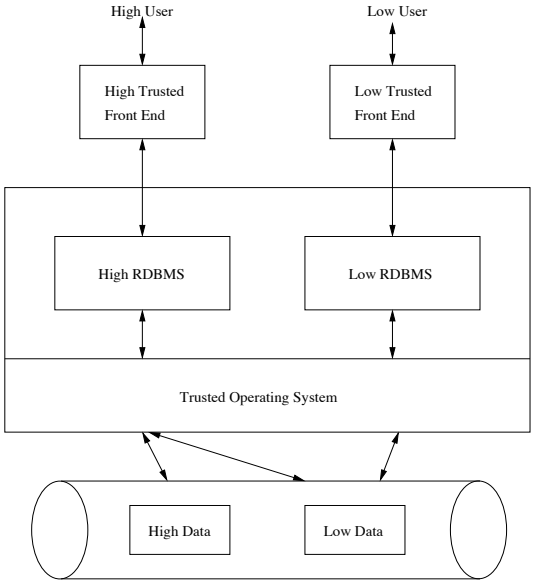


Figure 1: Multilevel secure kernelized RDBMS architecture.

3.1.2 Distributed Architectures

The distributed (or replicated) architecture[9, 10] is a variation of the kernelized architecture. It uses multiple copies of the trusted front-end and RDBMS, each associated with its own database storage. In this architecture scheme, an RDBMS at security level l contains a replica of every data item that a subject at level l can access. Thus, when data is retrieved, the RDBMS retrieves it only from its own database. Another benefit of this architecture is that data is physically separated into separate hardware databases. However, this scheme results in an additional overhead when data is updated as the various replicas need to be kept in sync.

3.2 Trusted Subjects Architectures

The trusted subject architecture[9] is a scheme that contains a trusted RDBMS and a trusted operating system. According to this architecture, the mandatory access control policy is enforced by the RDBMS itself. Database objects (*e.g.,* a table) are stored in operating system objects (*e.g.,* a file) labeled at the highest security level. A database table can con-

tain rows with different security levels. Such rows are distinguished based on their security level which is explicitly stored with each row. This architecture is called "trusted subject" because the RDBMS is *privileged* to violate the operating system's MAC policy when accessing database objects. For example, when a user with a low security level queries a database table, the operating system's object where that table is stored ends up being accessed, which is a violation of the operating system's MAC policy. But the RDBMS is trusted to return to the users only those rows for which he or she is authorized according to the MAC policy. Figure 2 illustrates a multilevel secure trusted subject RDBMS Architecture.

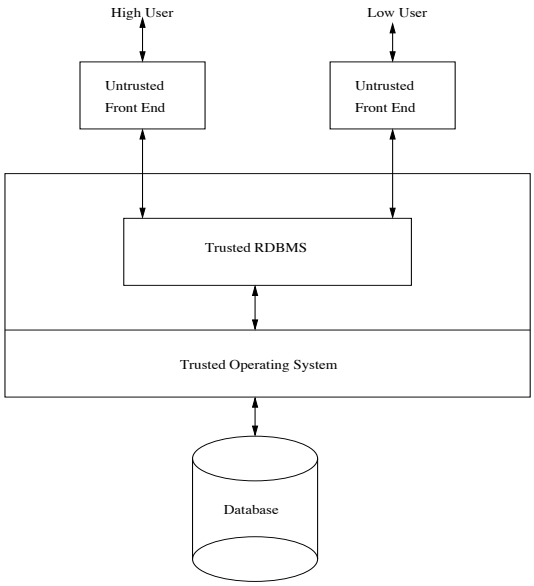


Figure 2: Multilevel secure trusted subject RDBMS architecture.

A benefit of this architecture is that the RDBMS has access to all levels of data at the same time, which minimizes retrieval and update processing. However, this architecture results in a special purpose RDBMS that requires a large amount of trusted code to be developed and verified along with the normal RDBMS features. It also lacks the potential to be evaluated to high TCSEC evaluation classes because meeting higher levels of assurance requires the ability to provide separation of mandatory objects by some form of hardware isolation. It is also difficult to prove that the trusted software used to isolate mandatory objects (*e.g.,* data rows with different security levels) is working correctly without allowing for the flow of data with high security level to users with low security level.

4 Polyinstantiation

Multilevel secure RDBMS utilize mandatory access control to prevent the unauthorized disclosure of high-level data to low-level users. It is also necessary to guard against the threat to confidentiality that can arise from enforcing database integrity constraints[6] across data from multiple security levels. To illustrate this threat to confidentiality, consider the following database table where the attribute "starship" is the primary key, and the attribute "label" represents the data row security level.

Starship	**Destination**	**Label**
Enterprise	Mars	High

Suppose that a user with a low security level wishes to insert the tuple (Enterprise, Talos, Low). From a purely database perspective, this insert must be rejected because it violates the primary key constraint. However, rejecting this insert could be sufficient to compromise security as the user with low security level could infer that the starship Enterprise is on a mission with a higher security level.

Polyinstantiation[8] is a solution to this problem. It expands the notion of primary key to include the security level so that more than one tuple may possess the same *apparent primary key* if they are at different security levels. To continue with our example, a new row with the same apparent primary key (*i.e.,* Enterprise) is added to the table.

Starship	**Destination**	**Label**
Enterprise	Mars	High
Enterprise	Talos	Low

From a security perspective, the newly added row is simply a *cover story* for the real mission of the starship enterprise.

In addition to protecting against inference, polyinstantiation is also useful to prevent denial of service to legitimate users as well as to protect against *storage covert channels*[5]. Covert channels use system variables and attributes to signal information. To illustrate this type of threat to confidentiality, consider the following database table where the attribute "starship" is the primary key, and the attribute "label" represents the data row security level.

Starship	Destination	Label
Enterprise	Talos	Low

Now, suppose that a user with a high security level wishes to update the destination to be "Mars". If the RDBMS rejects this update, then the user may have been denied legitimate privileges. If the update is allowed by changing the row's security level to "High" then a user with a low security level will notice that the data row has disappeared and will infer that its security level has been increased. If the update is allowed without changing the row's security level, then a storage covert channel will be created. That is, the data row itself could be used as a storage object for passing high level information to users with low security level. Polyinstantiation allows the RDBMS to insert a new data row with the same apparent primary key (*i.e.,* Enterprise) but with a high security level as a result of such update.

Starship	Destination	Label
Enterprise	Talos	Low
Enterprise	Mars	High

From a security perspective, the old data row is simply a cover story for the real mission of the starship enterprise.

5 Multilevel Secure Transaction Processing

Multilevel secure RDBMS utilize mandatory access control to prevent the unauthorized disclosure of high-level data to low-level users. It is also necessary to guard against the threat to confidentiality that can arise from employing conventional transaction protocols such as two-phase locking (2PL)[4]. The 2PL transaction protocol delays the execution of conflicting operations by setting locks on data items for read and write operations[6]. A transaction must acquire a shared-lock (S-lock) on a data item before reading it and an exclusive lock (X-lock) before writing it. The 2PL transaction protocol is inherently vulnerable to timing covet channels which could be established to leak confidential information. A timing covert channel [5] varies the amount of time to complete a task to signal information. To illustrate this threat to confidentiality, consider the following example.

Let T_i denote a high security level transaction, which is reading a low security level data item A. Let T_j denote a low security level transaction, which is trying to write to data item A. If the 2PL transaction protocol is employed, then T_j must wait to acquire an X-lock on data item A (*i.e.,* wait until T_i releases its S-lock on data item A). Suppose that T_j can measure the time quantum q it has to wait to acquire the lock on data item A: A quantum of waiting time greater than a certain amount represents '1', and a quantum of waiting time less than that a certain amount represents '0'. Transaction T_i can exploit this knowledge to send one bit of high security level information to T_j, and by repeating this protocol, any information can be sent, creating a timing covert channel.

2PL, and in general conventional transaction protocols in RDBMS, are not secure against timing covert channels.

6 Commercial and Research Multilevel Secure RDBMS

The research and development efforts in the area of multilevel secure RDBMS have resulted in a number of commercial and research systems. The most noticeable of these systems are the following: Trusted Oracle[12], Informix OnLine/Secure[13], Sybase Secure SQL Server[14], DB2 for z/OS[15], Trusted

Rubix[16], SEAVIEW[8], and Unisys Secure Distributed DBMS[17].

Trusted Oracle can be configured to run in one of two modes: DBMS MAC and OS MAC. The former is an architecture where mandatory access control is enforced by the RDBMS itself, and thus is a trusted subject architecture. The latter is a kernelized architecture (*i.e.,* mandatory access control is delegated to the operating system). Informix OnLine/Secure, Sybase Secure SQL Server, DB2 for z/OS, and Trusted Rubix are examples of a trusted subject architecture. The SEAVIEW research system is an example of a kernelized architecture whereas the Unisys Secure Distributed research RDBMS is an example of a distributed architecture.

Informix OnLine/Secure, Sybase Secure SQL Server, Trusted Oracle, and Trusted Rubix all support polyinstantiation. The key for a tuple in Informix OnLine/Secure automatically includes the tuple security label. Thus, polyinstantiation is always possible and cannot be suppressed by the RDBMS.

The tuple security label in the Sybase Secure SQL Server is part of all keys. Thus, polyinstantiation is always possible and cannot be suppressed by the RDBMS.

Trusted Oracle can be configured to run in one of two modes. When run in DBMS MAC mode, a single Trusted Oracle database can store information at multiple security levels. In this mode, Trusted Oracle can turn polyinstantiation on and off at the table level by requiring key integrity which does not include the tuple security label. When on, the primary key includes the tuple label, which allows polyinstantiation to occur. When off, the key does not include the tuple security label, thus preventing polyinstantiation.

When run in OS MAC mode, Trusted Oracle is capable of storing data at only a single security label, and the RDBMS is constrained by the underlying operating system MAC policy. Without any MAC privilege, the RDBMS cannot suppress polyinstantiation because a low RDBMS will not be aware of any tuple with the same primary key at a higher security level, and a high RDBMS cannot be trusted to modify data at a low security level. As such, polyinstantiation cannot be prevented when Trusted Oracle is running in OS MAC mode.

Informix OnLine/Secure and Trusted Oracle provide secure transaction processing protocols. Informix OnLine/Secure uses an approach by which a transaction at a low security level can acquire a write lock on a low data item even if a transaction at a high security level holds a read lock on that data item. Thus, a transaction at a low security level is never delayed by a transaction at a high security level. The transaction at the high security level simply receives a warning that a lock on a low data item has been "broken". Trusted Oracle uses a combination of locking and multiversioning techniques.

7 Conclusion

This paper has given an overview of multilevel security, the MLS evaluation and certification process, and multilevel secure RDBMS. Building a multilevel secure RDBMS can be a challenging task. Depending on the architecture followed, this might require rebuilding major portions of an existing commercial RDBMS. It also requires significant effort to evaluate and certify, particularly if a high level of assurance is sought. We are not aware of any commercial RDBMS that has been evaluated higher than B1 according to the Trusted Computer Security Evaluation Criteria.

Mandatory access control, polyinstantiation, and secure transaction processing are the key aspects of a multilevel secure RDBMS. However, these are not sufficient to ensure that security cannot be compromised. Depending on how stringent the requirements of the organization that wishes to deploy a multilevel secure RDBMS, the RDBMS might have to implement additional security guards. For example, SQL compilers have traditionally been guided by performance reasons in selecting the order in which the predicates contained in a query are evaluated (*i.e.,* more selective predicates are often evaluated first to narrow down the set of rows to be passed on to a subsequent join because join operations are costly). If the method chosen to enforce MAC when accessing a table is based on query modification to incorporate the MLS two security properties in

the form of regular predicates, then special care must be taken in selecting the order in which the predicates on that table are evaluated to avoid unauthorized leakage of data rows. To illustrate how leakage could occur, suppose that a query has a predicate on a table that involves a User-Defined Function (UDF). Further suppose that this UDF takes the whole data row as an input parameter and that the UDF source code makes a copy of the data row outside the database (or sends it as an e-mail to some destination). Now, assume that some data row R cannot be returned to the user who issued the query because this would violate the MLS security properties. If the predicate involving the UDF is evaluated prior to evaluating the predicates that implement the MLS security properties then data row R will be consumed by the UDF and consequently leaked to an unauthorized user.

Database triggers[6] are another example where additional security guards could be necessary. A trigger could cause labeled data row to flow from a table on which mandatory access control is enforced to another table on which mandatory access control is not enforced. Without proper flow control measures, triggers could cause unauthorized leakage of information to occur.

Acknowledgements

The author wishes to thank Calisto Zuzarte and Kelly Lyons from the IBM Toronto Laboratory for their suggestion to write a CASCON paper about multilevel secure RDBMS.

Trademarks

Disclaimer

The views expressed in this paper are those of the authors and not necessarily of IBM Canada Ltd. or IBM Corporation.

About the Author

Walid Rjaibi joined IBM in 1996. He initially worked at the IBM Toronto Lab within the DB2 UDB query optimization team for five years. In this role, Walid was the architect and author of several innovative solutions including the extensions made to the DB2 statistics model to support parallel database environments and the query performance simulator. Walid then joined IBM Research in Zurich, Switzerland (ZRL) where he worked as a Research Staff Member in Network Security and Cryptography for two years. At ZRL, he was an active member of the IBM Privacy Technology Institute (PTI) where he developed innovative solutions for enabling RDBMS to automatically enforce privacy policies. Walid returned to the IBM Toronto Lab in March 2003 where he joined the newly formed DB2 UDB security development team. He has authored several research and technical papers on database security and privacy, and holds a patents portfolio of eleven filed or granted patents. Walid holds a Computer Engineer degree from the University of Tunis (Tunisia), and a Masters degree in Computer Science from Laval University (Canada).

References

[1] W. Rjaibi, P. Bird. A Multi-Purpose Implementation of Mandatory Access Control in Relational Database Management Systems. In *Proc. of the 30th International Conference on Very Large Databases*, Toronto, Canada, 2004.

[2] Department of Defense. Multilevel Security in the Department Of Defense: The Basics. http://nsi.org/Library/Compsec/sec0.html.

[3] The official website of the Common Criteria Project http://www.commoncriteriaportal.org/

[4] V. Atluri, S. Jajodia, T. F. Keefe, C. MaCollum, R. Mukkamal. Multilevel Secure Transaction Processing: Status and Prospects. *Database Security, X: Status and Prospects*, Chapman & Hall 1997, eds. Pierangela Samarati and Ravi Sandhu.

[5] Trusted Computer Security Evaluation Criteria, DoD 5200.28-STD. U.S. Department of Defense, 1985.

[6] R. Elmasri, S. Navathe. Fundamentals of Database Systems. ISBN 0-201-54263-3, Addison-Wesley, 2000.

[7] S. Jajodia, R. Sandhu. Toward a Multilevel Secure Relational Data Model. In *Proc. of ACM SIGMOD*, Denver, Colorado, USA, 1991.

[8] D. E. Denning. The Sea View Security Model. In *Proc. of the IEEE Symposium on Security and Privacy*, Oakland, California, USA, 1988.

[9] M. D. Abrams, S. Jajodia, H. J. Podell. Information Security An Integrated Collection of Essays. *IEEE Computer Society Press*, Los Alamitos, CA, USA, 1995.

[10] S. Castano, et al. Database Security. *ACM Press*, New York, NY, USA, 1995.

[11] E. Bell, L. J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report MTR-2997, The Mitre Corporation, Burlington Road, Bedford, MA 01730, USA.

[12] Oracle Corporation. Trusted Oracle Administrator's Guide. Redwood City, CA, USA, 1992.

[13] Informix. Informix OnLine/Secure Administrator's Guide. Menlo Park, CA, USA, 1993.

[14] Sybase Inc. Building Applications for Secure SQL Server, Sybase Secure SQL Server Release 10.0. Emeryville, CA, USA, 1993.

[15] IBM Corporation. DB2 UDB for z/OS V8 Administration Guide. 2004.

[16] National Computer Security Center. Polyinstantaition Issues in Multilevel Secure Relational Database Management Systems. NCSC Technical Report - 005, Volume 3/5, Library No. S-243,039, May 1996.

[17] LouAnna Notargiacomo. Architectures for MLS Database Management Systems. Information Security:An Integrated Collection of Essays, IEEE Computer Society Press, Los Alamitos, California, USA.

Table C.2 – Granted Patents

ID	Publication	Key Contributions
1	**Controlling Data Access Using Security Label Components** *US Patent US7,568,235B2*	This patent is the foundation for the security label concepts discussed in the core publication #1 in table C.1 above.
2	**Method for Modifying a Query by Use of an External System for Managing Assignments of User and Data Classifications** *US Patent US7,860,875B2*	This patent is the foundation for the enterprise integration methodology discussed in the core publication #1 in table C.1 above.
3	**Fine-Grained, Label-Based, XML Access Control Model** *US Patent US2009/0063951A1*	This patent is the foundation for the inter-node relationship labelling concept discussed in the publication #2 in table C.1 above.

US007568235B2

(12) **United States Patent**
Bird et al.

(10) **Patent No.: US 7,568,235 B2**
(45) **Date of Patent: Jul. 28, 2009**

(54) **CONTROLLING DATA ACCESS USING SECURITY LABEL COMPONENTS**

(75) Inventors: **Paul Miller Bird**, Markham (CA); **Walid Rjaibi**, Markham (CA)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 828 days.

(21) Appl. No.: **11/036,839**

(22) Filed: **Jan. 15, 2005**

(65) **Prior Publication Data**

US 2006/0059567 A1 Mar. 16, 2006

(30) **Foreign Application Priority Data**

Feb. 20, 2004 (CA) 2459004

(51) **Int. Cl.**

H04L 9/32	(2006.01)
H04L 9/00	(2006.01)
G06F 17/30	(2006.01)
G06F 7/04	(2006.01)
H04K 1/00	(2006.01)

(52) **U.S. Cl.** **726/27**; 726/6; 726/28; 713/182

(58) **Field of Classification Search** 726/6, 726/27, 28; 713/182

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,185,551	B1	2/2001	Birrell et al.	707/3
6,526,398	B2	2/2003	Wolff et al.	707/1
6,606,681	B1	8/2003	Uzun	711/108
6,981,265	B1 *	12/2005	Rees et al.	719/313
2001/0013096	A1	8/2001	Luckenbaugh et al.	713/154
2004/0015701	A1 *	1/2004	Flyntz	713/182

* cited by examiner

Primary Examiner—Kambiz Zand
Assistant Examiner—Aubrey H Wyszynski
(74) *Attorney, Agent, or Firm*—Patterson & Sheridan, LLP

(57) **ABSTRACT**

A method that controls user access to the stored data elements using security label components is disclosed. Each stored data element is associated with a set of data security label components, and each user is associated with a set of user security label components. The method receives a user request to access the stored data elements, compares the set of user security label components to the set of data security label components associated with the users, and based on the comparison result, determines whether or not to permit access to the stored data.

19 Claims, 14 Drawing Sheets

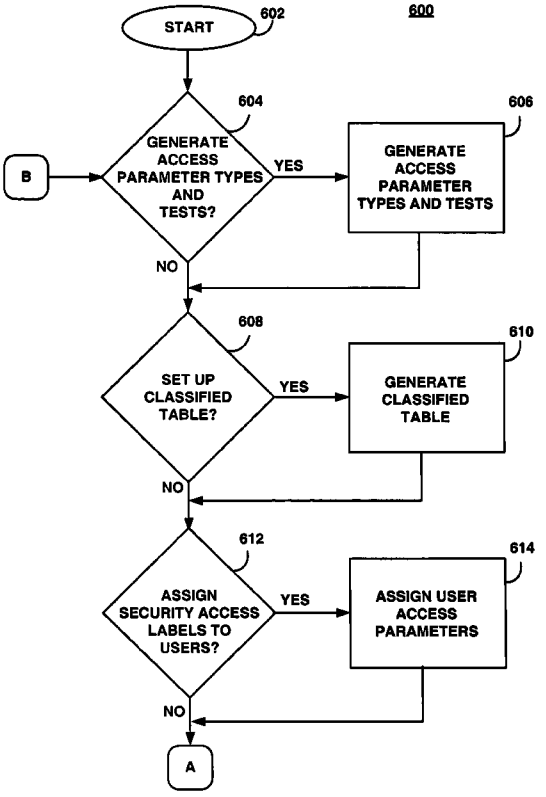


FIG. 1

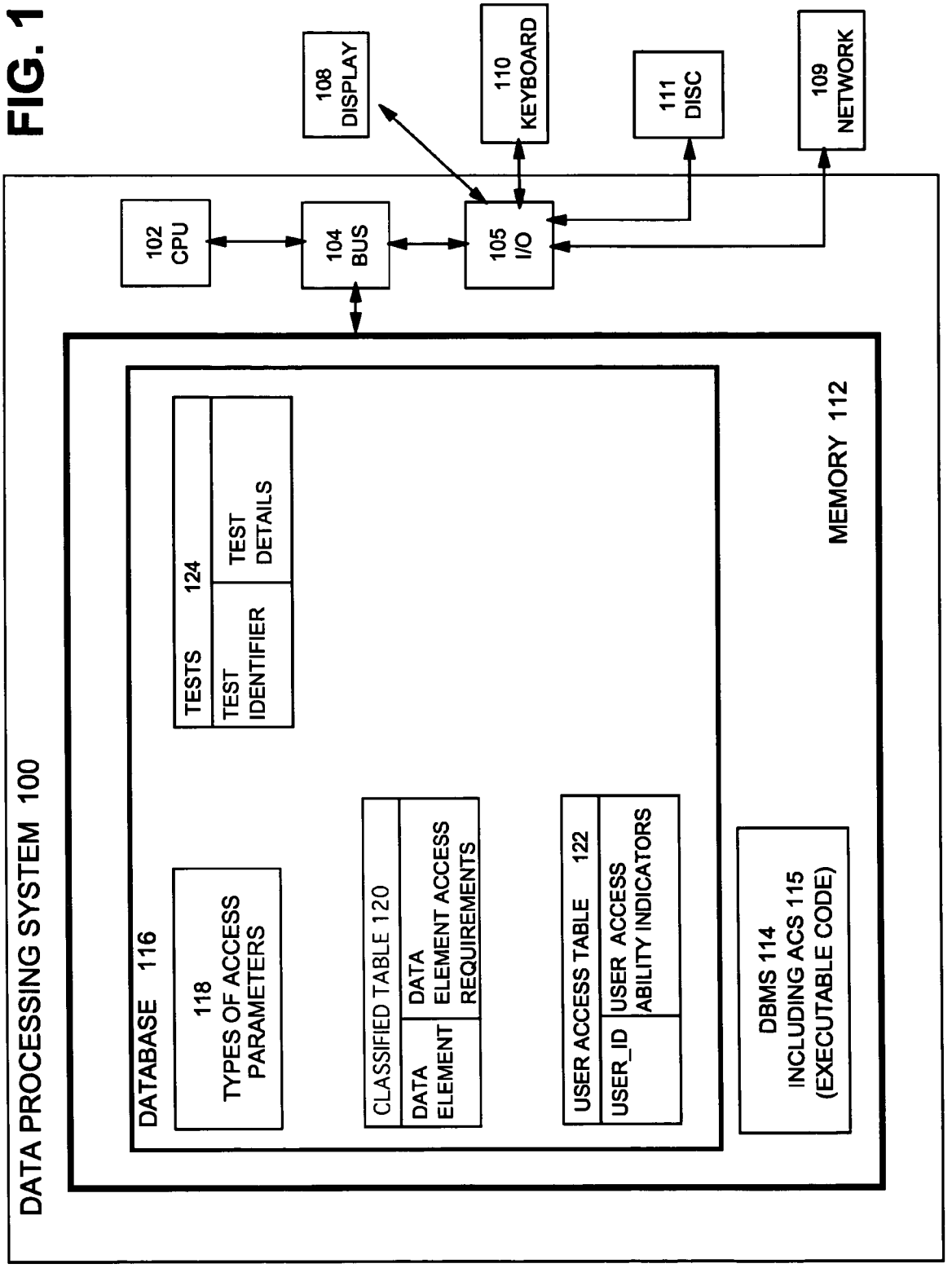

DATA PROCESSING SYSTEM 100
102 CPU
104 BUS
105 I/O
108 DISPLAY
110 KEYBOARD
111 DISC
109 NETWORK
MEMORY 112
DATABASE 116
118 TYPES OF ACCESS PARAMETERS
TESTS 124
TEST IDENTIFIER
TEST DETAILS
CLASSIFIED TABLE 120
DATA ELEMENT
DATA ELEMENT ACCESS REQUIREMENTS
USER ACCESS TABLE 122
USER_ID
USER ACCESS ABILITY INDICATORS
DBMS 114 INCLUDING ACS 115 (EXECUTABLE CODE)

FIG. 2

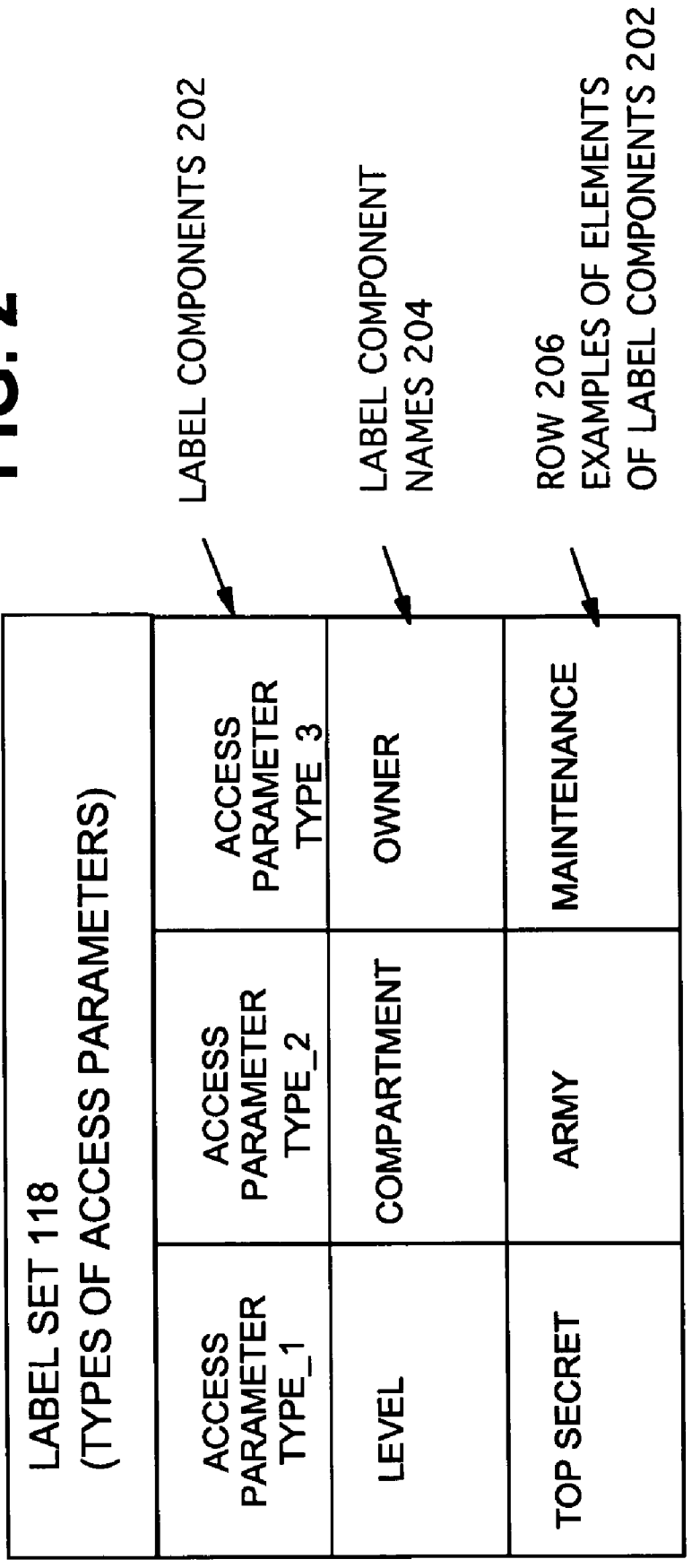

LABEL SET 118
(TYPES OF ACCESS PARAMETERS)
ACCESS PARAMETER TYPE_1
ACCESS PARAMETER TYPE_2
ACCESS PARAMETER TYPE_3
LEVEL
COMPARTMENT
OWNER
TOP SECRET
ARMY
MAINTENANCE
LABEL COMPONENTS 202
LABEL COMPONENT NAMES 204
ROW 206 EXAMPLES OF ELEMENTS OF LABEL COMPONENTS 202

FIG.3

CLASSIFIED TABLE 120 (TABLE 120 CONTAINING CLASSIFIED DATA ELEMENTS)				
302 DATA ELEMENT IDENTIFIER	303 CLASSIFIED DATA ELEMENT	304 DATA ELEMENT ACCESS REQUIREMENTS TO BE COMPARED AGAINST USER ACCESSIBILITY INDICATORS		
		ROW LABELS 306 DATA ELEMENT ACCESS COMPONENT TYPE_1 (LEVEL)	ROW LABELS 308 DATA ELEMENT ACCESS REQUIREMENT TYPE_2 (COMPARTMENT)	ROW LABELS 310 DATA ELEMENT ACCESS REQUIREMENT TYPE_3 (OWNER)
1	PLAN_A	TOP SECRET	ARMY	MARINES
2	PLAN_B	SECRET	NASA	RESEARCH
3	PLAN_C	CLASSIFIED	AIR FORCE	MAINTENANCE
4	PLAN_D	DECLASSIFIED	EMBASSY	LIBRARY

312
314
316
318

FIG. 4

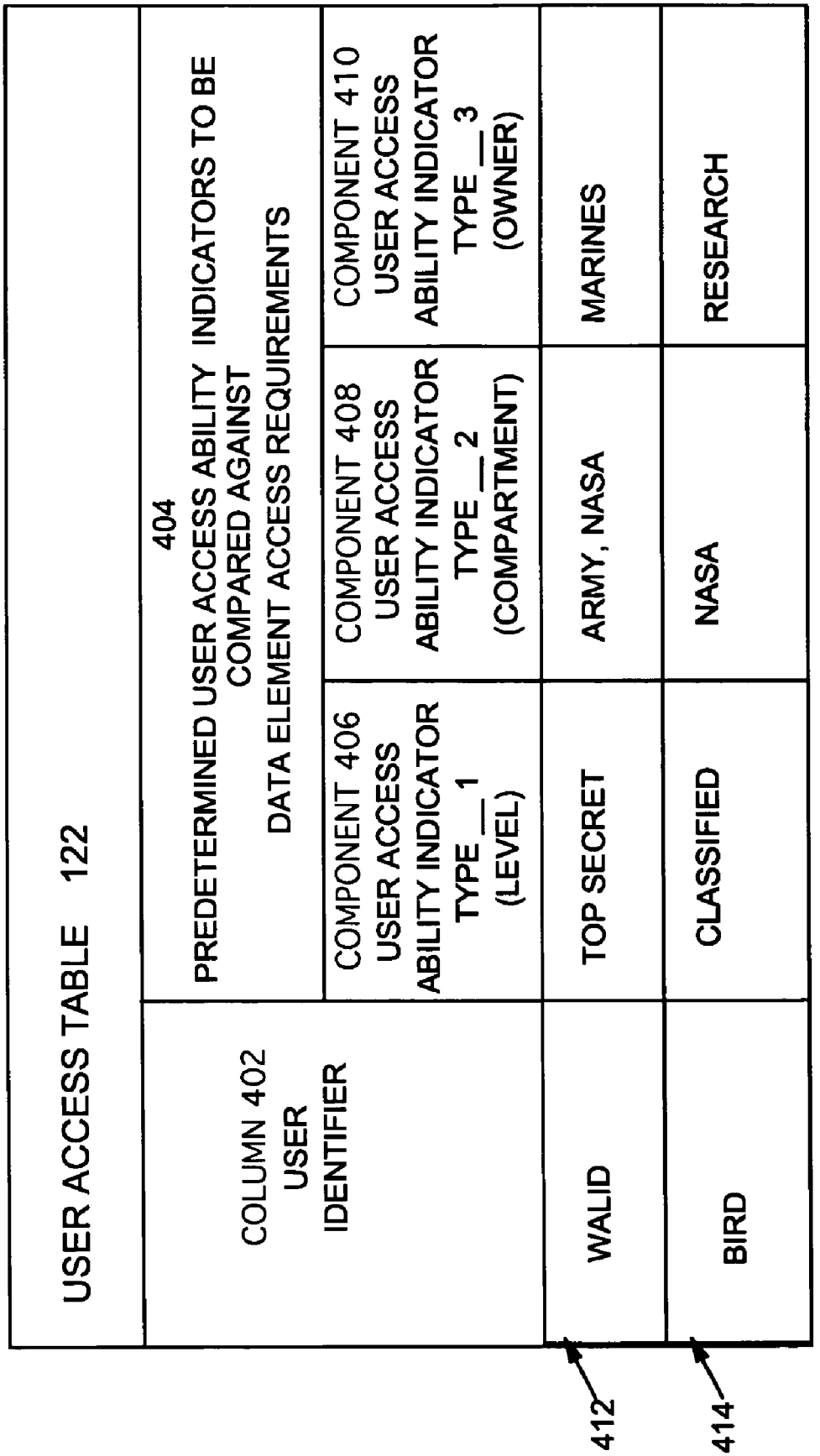
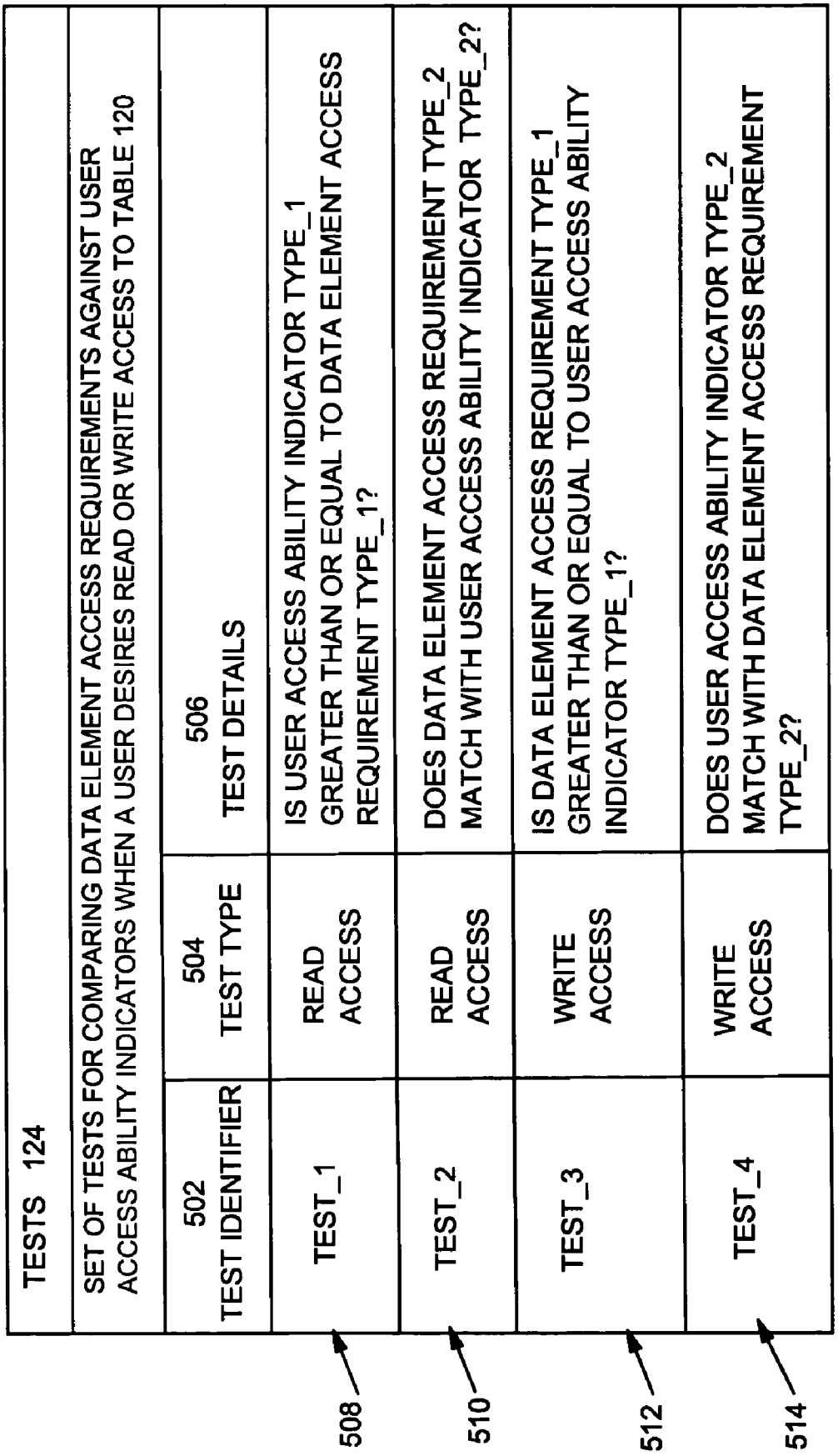

USER ACCESS TABLE 122
COLUMN 402
USER
IDENTIFIER
404
PREDETERMINED USER ACCESS ABILITY INDICATORS TO BE COMPARED AGAINST
DATA ELEMENT ACCESS REQUIREMENTS
COMPONENT 406
USER ACCESS
ABILITY INDICATOR
TYPE__1
(LEVEL)
COMPONENT 408
USER ACCESS
ABILITY INDICATOR
TYPE__2
(COMPARTMENT)
COMPONENT 410
USER ACCESS
ABILITY INDICATOR
TYPE__3
(OWNER)
WALID
TOP SECRET
ARMY, NASA
MARINES
BIRD
CLASSIFIED
NASA
RESEARCH
412
414

FIG. 5


TESTS 124
SET OF TESTS FOR COMPARING DATA ELEMENT ACCESS REQUIREMENTS AGAINST USER ACCESS ABILITY INDICATORS WHEN A USER DESIRES READ OR WRITE ACCESS TO TABLE 120
502 TEST IDENTIFIER
504 TEST TYPE
506 TEST DETAILS
TEST_1
READ ACCESS
IS USER ACCESS ABILITY INDICATOR TYPE_1 GREATER THAN OR EQUAL TO DATA ELEMENT ACCESS REQUIREMENT TYPE_1?
TEST_2
READ ACCESS
DOES DATA ELEMENT ACCESS REQUIREMENT TYPE_2 MATCH WITH USER ACCESS ABILITY INDICATOR TYPE_2?
TEST_3
WRITE ACCESS
IS DATA ELEMENT ACCESS REQUIREMENT TYPE_1 GREATER THAN OR EQUAL TO USER ACCESS ABILITY INDICATOR TYPE_1?
TEST_4
WRITE ACCESS
DOES USER ACCESS ABILITY INDICATOR TYPE_2 MATCH WITH DATA ELEMENT ACCESS REQUIREMENT TYPE_2?
508
510
512
514

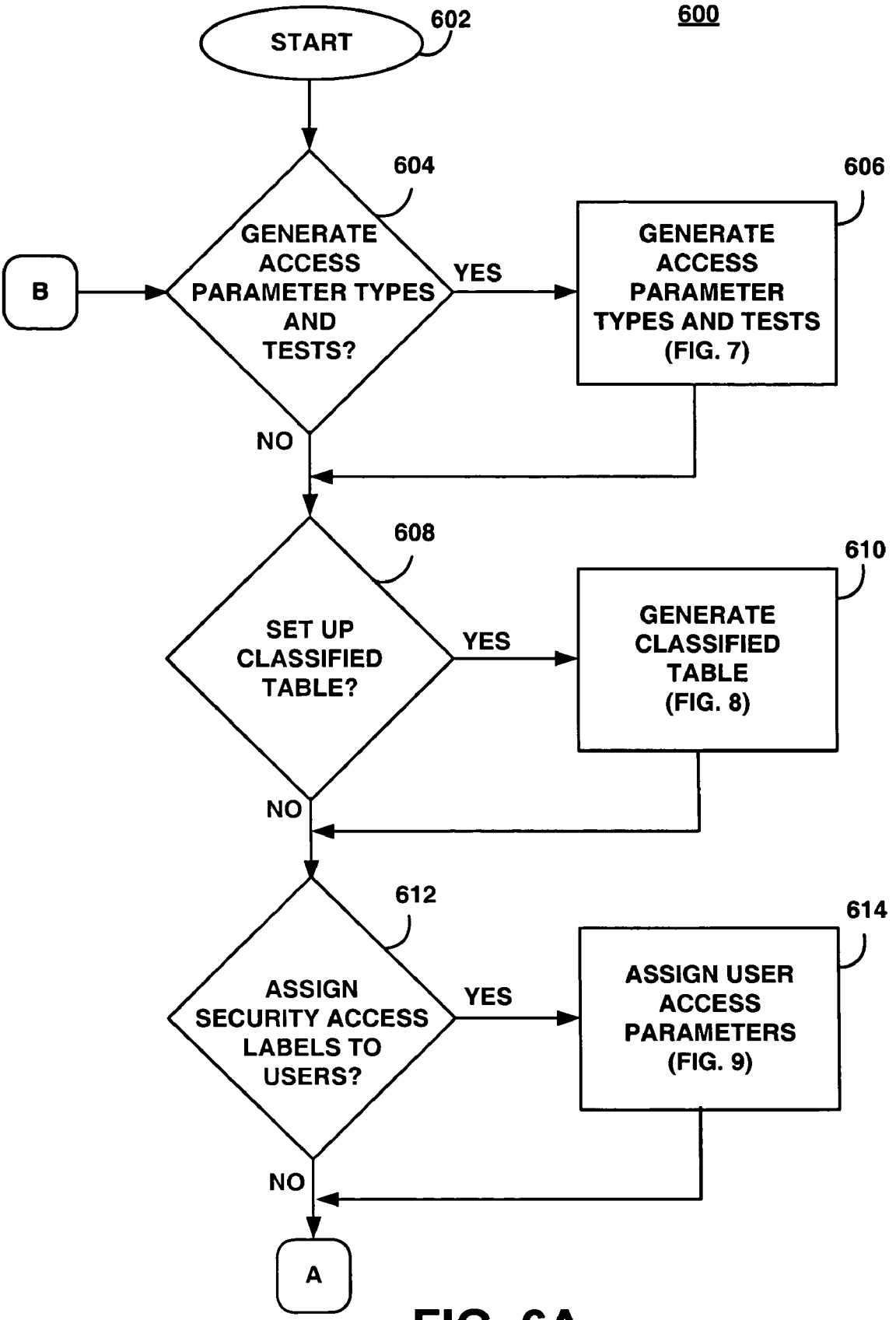

600
START
602
B
GENERATE ACCESS PARAMETER TYPES AND TESTS?
604
YES
NO
GENERATE ACCESS PARAMETER TYPES AND TESTS (FIG. 7)
606
SET UP CLASSIFIED TABLE?
608
YES
NO
GENERATE CLASSIFIED TABLE (FIG. 8)
610
ASSIGN SECURITY ACCESS LABELS TO USERS?
612
YES
NO
ASSIGN USER ACCESS PARAMETERS (FIG. 9)
614
A

FIG. 6A

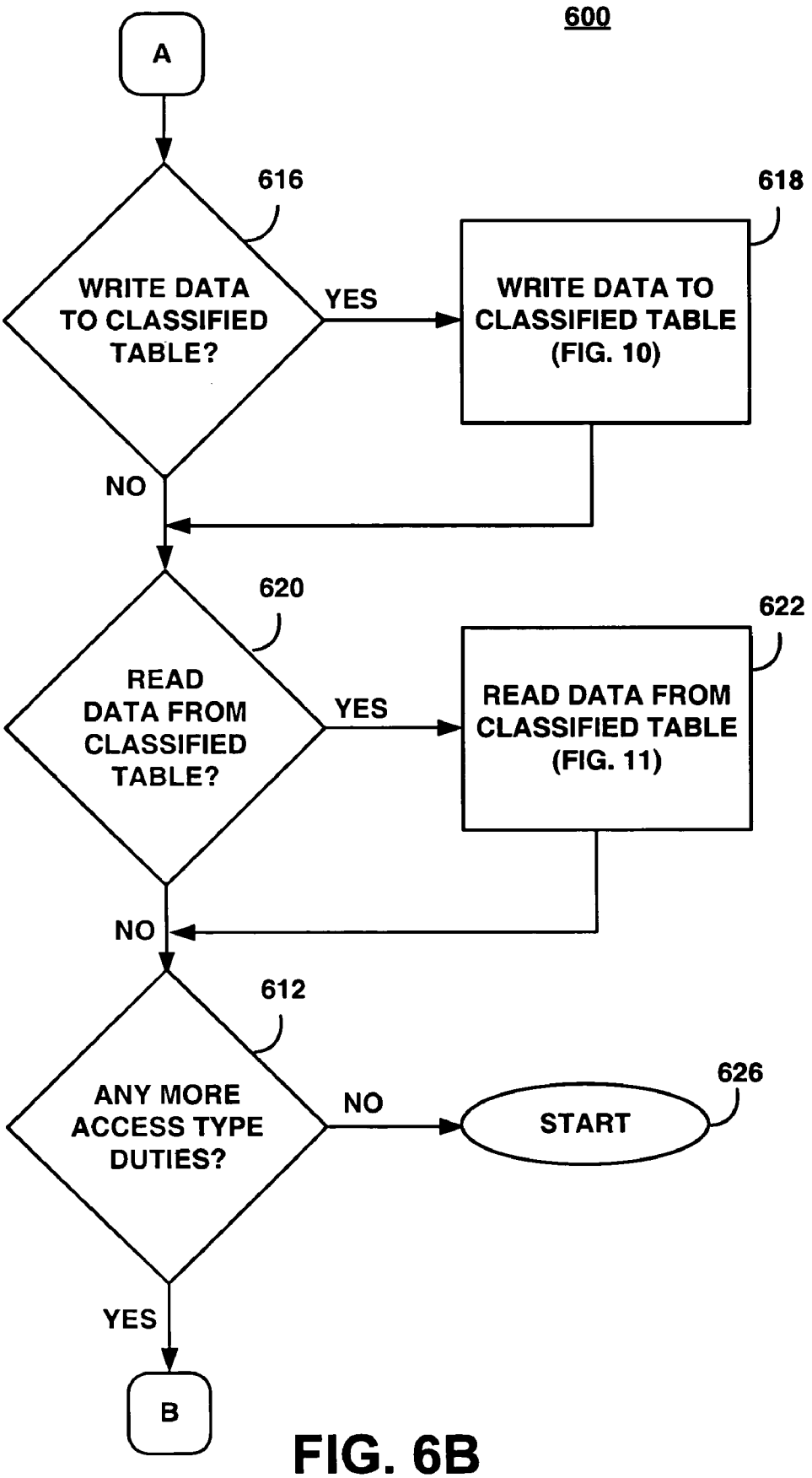

600
A
616
WRITE DATA TO CLASSIFIED TABLE?
YES
618
WRITE DATA TO CLASSIFIED TABLE (FIG. 10)
NO
620
READ DATA FROM CLASSIFIED TABLE?
YES
622
READ DATA FROM CLASSIFIED TABLE (FIG. 11)
NO
612
ANY MORE ACCESS TYPE DUTIES?
NO
626
START
YES
B

FIG. 6B

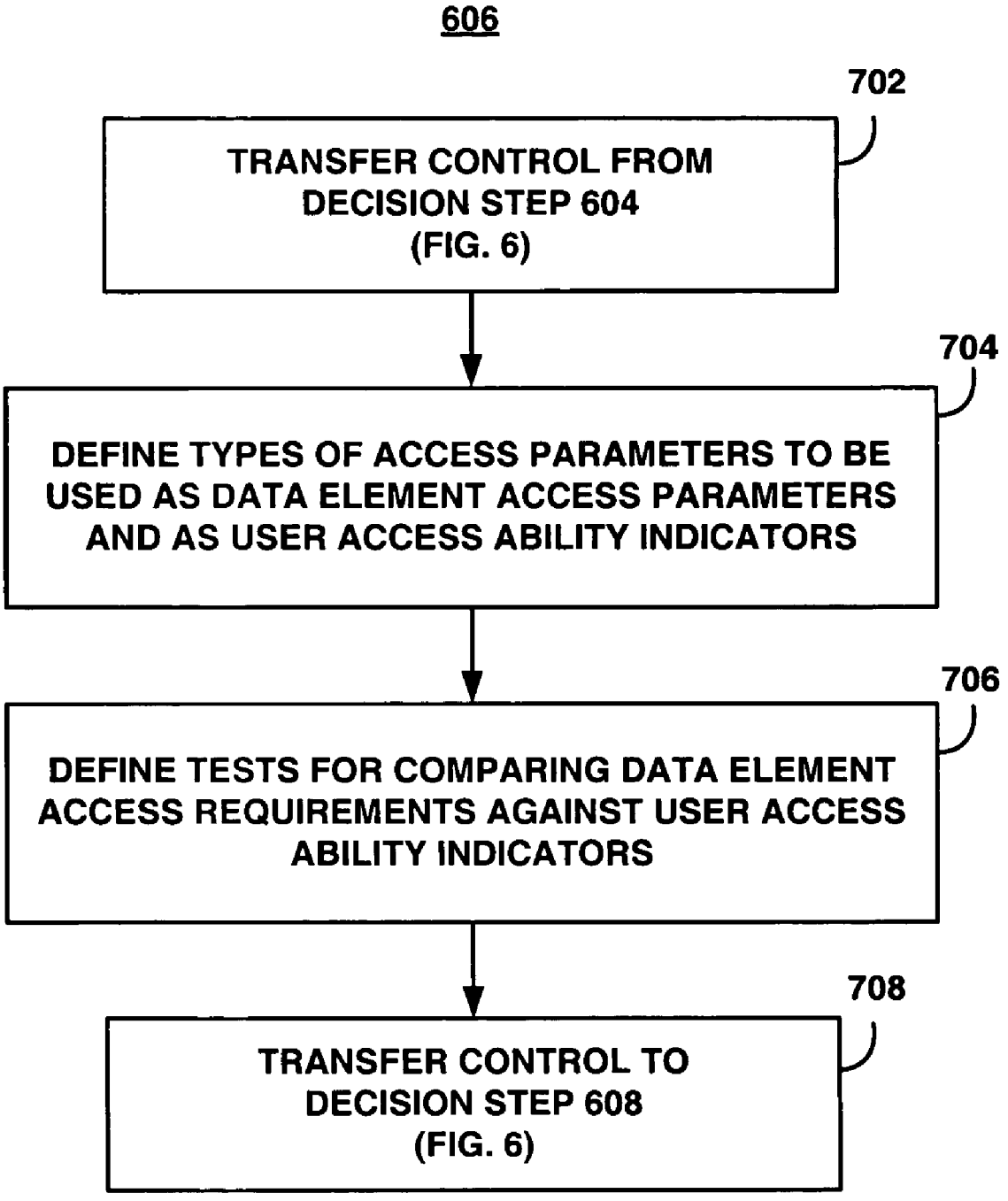

606
702
TRANSFER CONTROL FROM DECISION STEP 604 (FIG. 6)
704
DEFINE TYPES OF ACCESS PARAMETERS TO BE USED AS DATA ELEMENT ACCESS PARAMETERS AND AS USER ACCESS ABILITY INDICATORS
706
DEFINE TESTS FOR COMPARING DATA ELEMENT ACCESS REQUIREMENTS AGAINST USER ACCESS ABILITY INDICATORS
708
TRANSFER CONTROL TO DECISION STEP 608 (FIG. 6)

FIG. 7

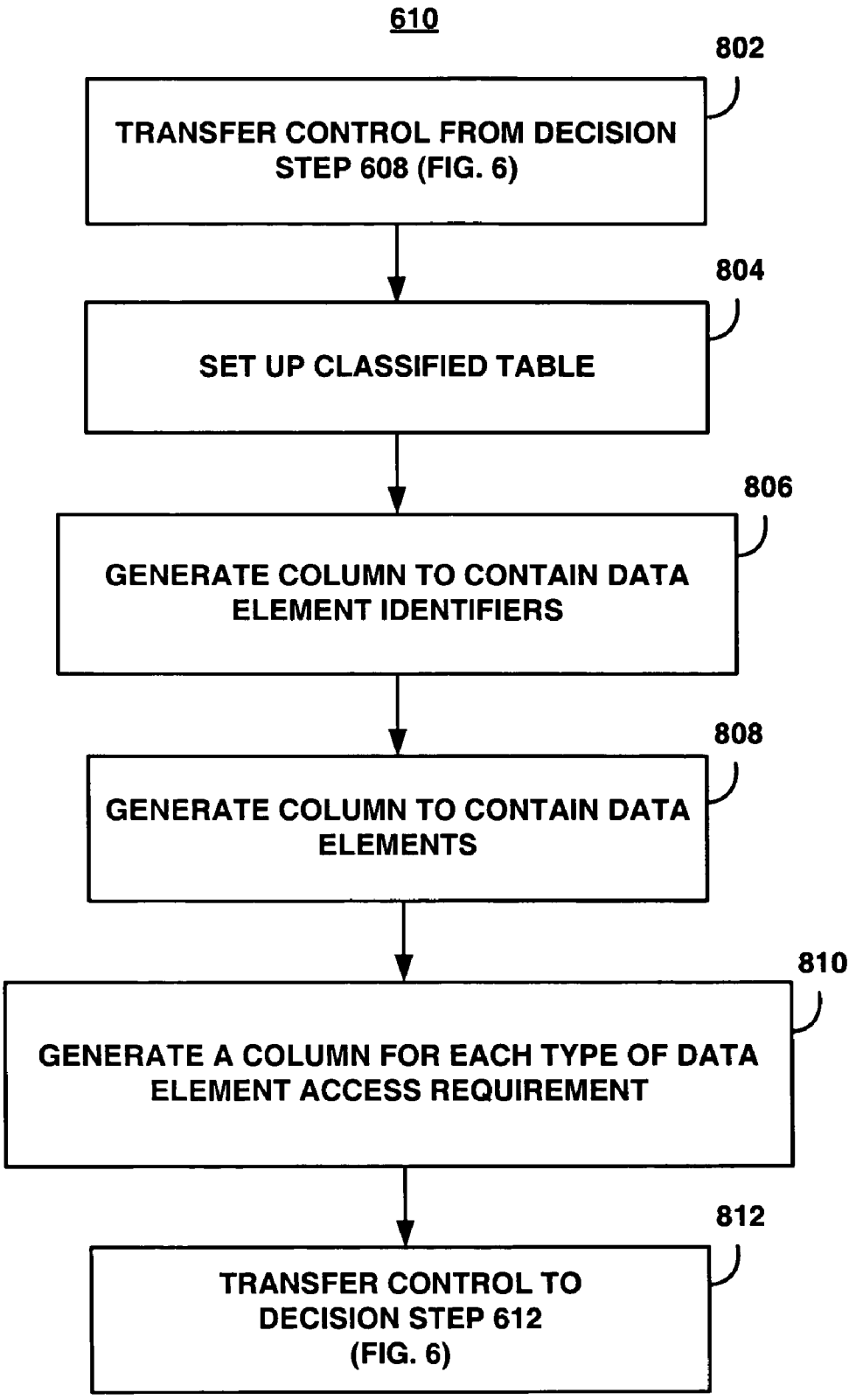

610
802
TRANSFER CONTROL FROM DECISION STEP 608 (FIG. 6)
804
SET UP CLASSIFIED TABLE
806
GENERATE COLUMN TO CONTAIN DATA ELEMENT IDENTIFIERS
808
GENERATE COLUMN TO CONTAIN DATA ELEMENTS
810
GENERATE A COLUMN FOR EACH TYPE OF DATA ELEMENT ACCESS REQUIREMENT
812
TRANSFER CONTROL TO DECISION STEP 612 (FIG. 6)

FIG. 8

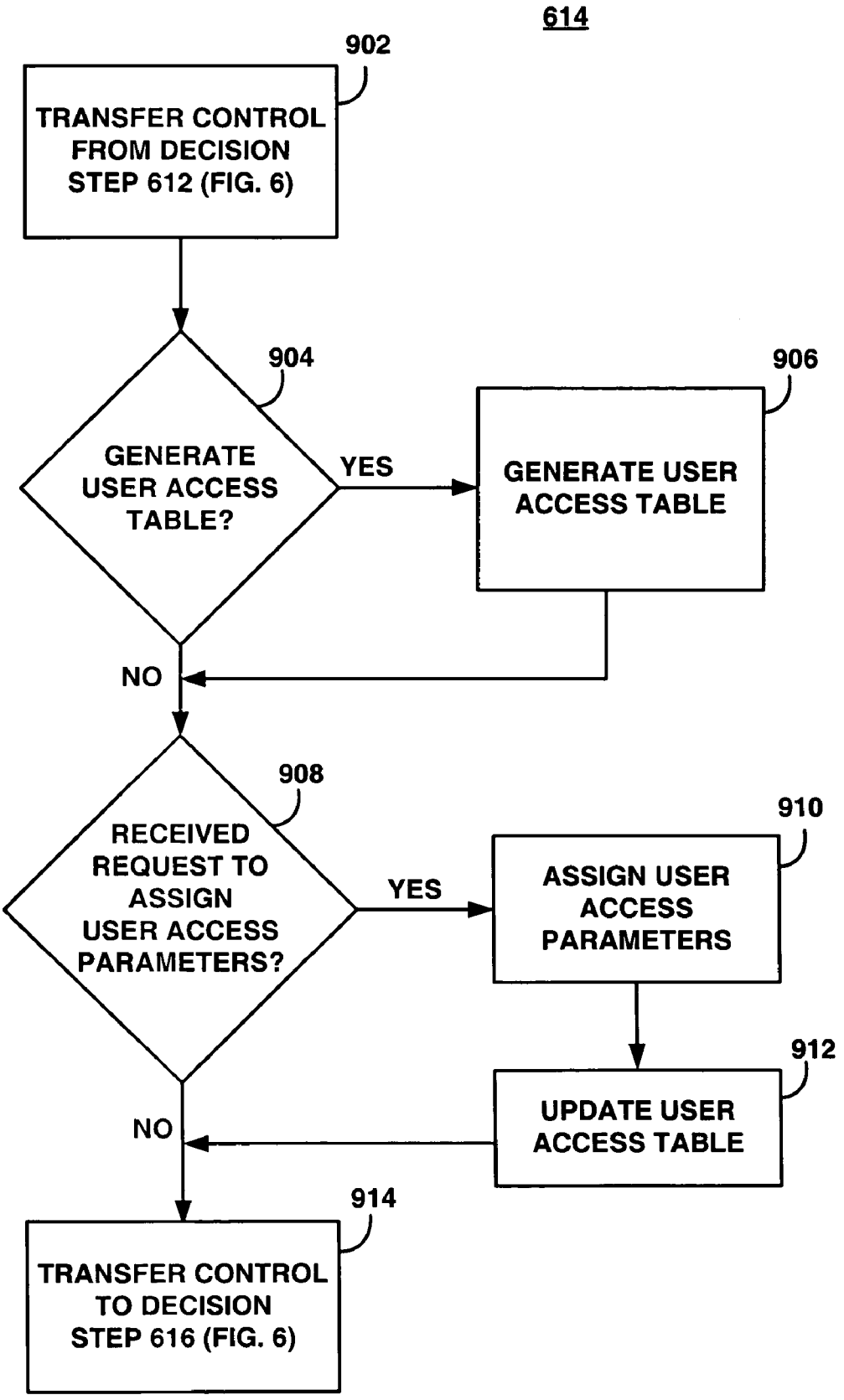

614
902
TRANSFER CONTROL FROM DECISION STEP 612 (FIG. 6)
904
GENERATE USER ACCESS TABLE?
YES
906
GENERATE USER ACCESS TABLE
NO
908
RECEIVED REQUEST TO ASSIGN USER ACCESS PARAMETERS?
YES
910
ASSIGN USER ACCESS PARAMETERS
912
UPDATE USER ACCESS TABLE
NO
914
TRANSFER CONTROL TO DECISION STEP 616 (FIG. 6)

FIG. 9

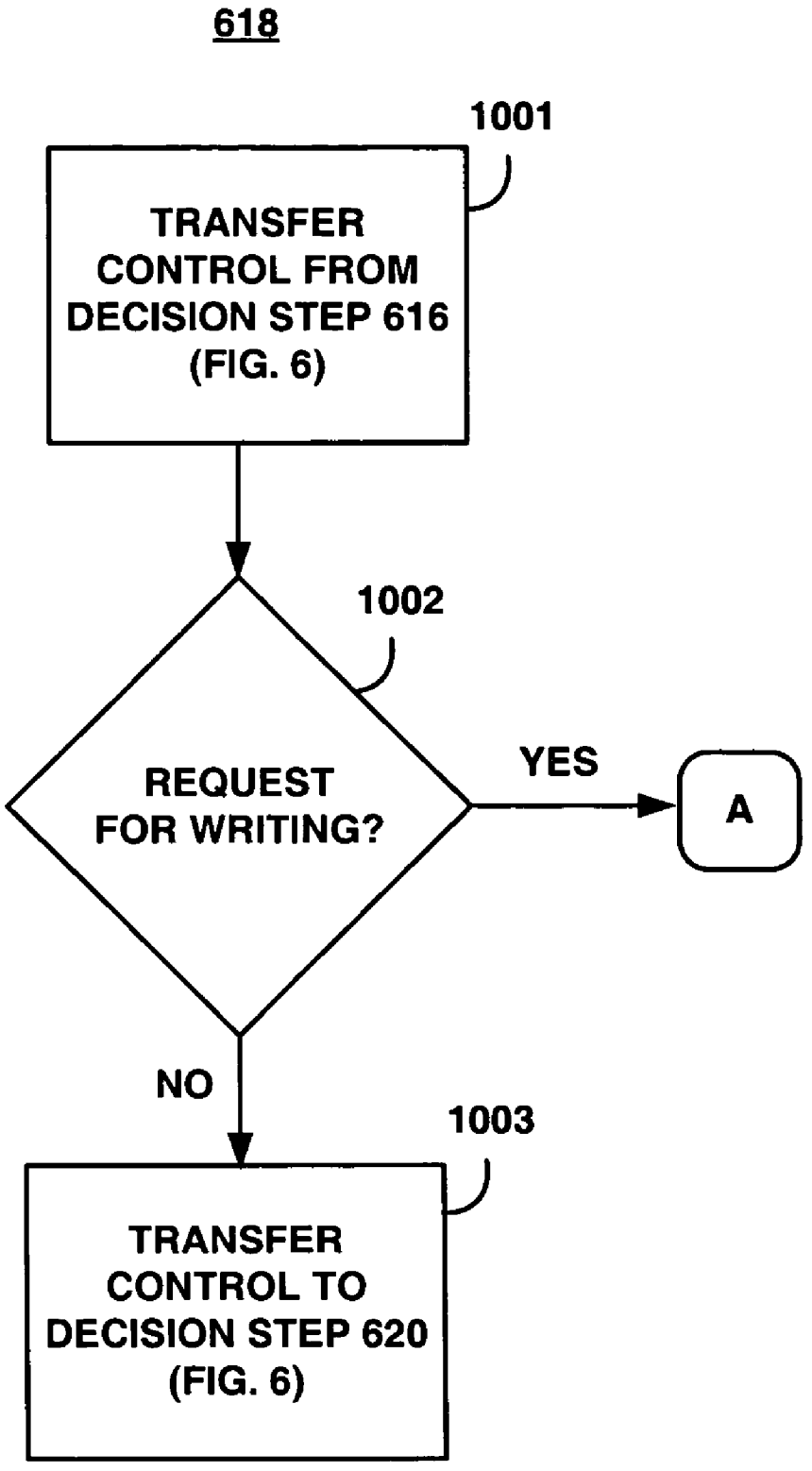

618
1001
TRANSFER CONTROL FROM DECISION STEP 616 (FIG. 6)
1002
REQUEST FOR WRITING?
YES
A
NO
1003
TRANSFER CONTROL TO DECISION STEP 620 (FIG. 6)

FIG. 10A

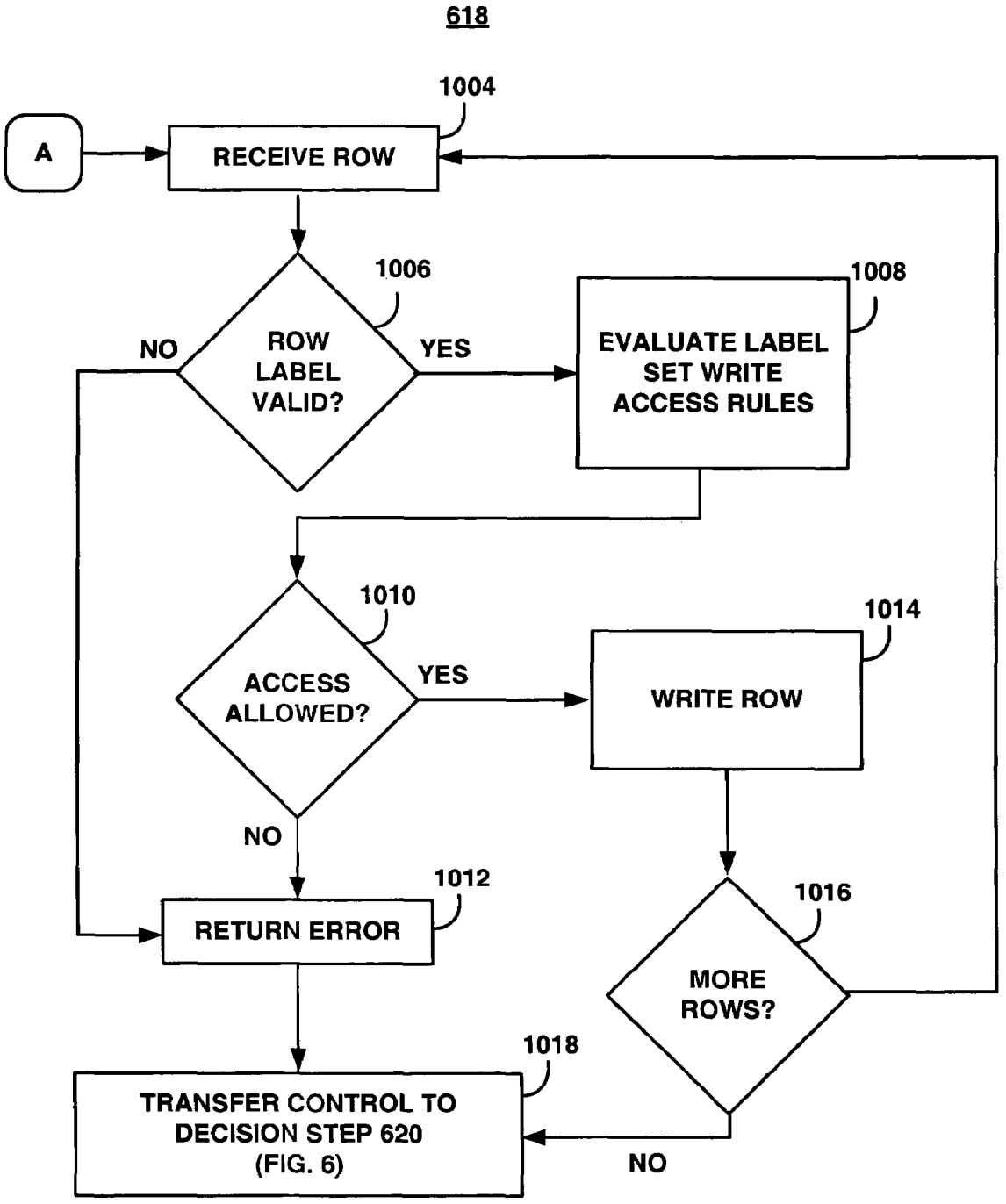

618
1004
A
RECEIVE ROW
1006
ROW LABEL VALID?
NO
YES
1008
EVALUATE LABEL SET WRITE ACCESS RULES
1010
ACCESS ALLOWED?
YES
1014
WRITE ROW
NO
1012
RETURN ERROR
1016
MORE ROWS?
1018
TRANSFER CONTROL TO DECISION STEP 620 (FIG. 6)
NO

FIG. 10B

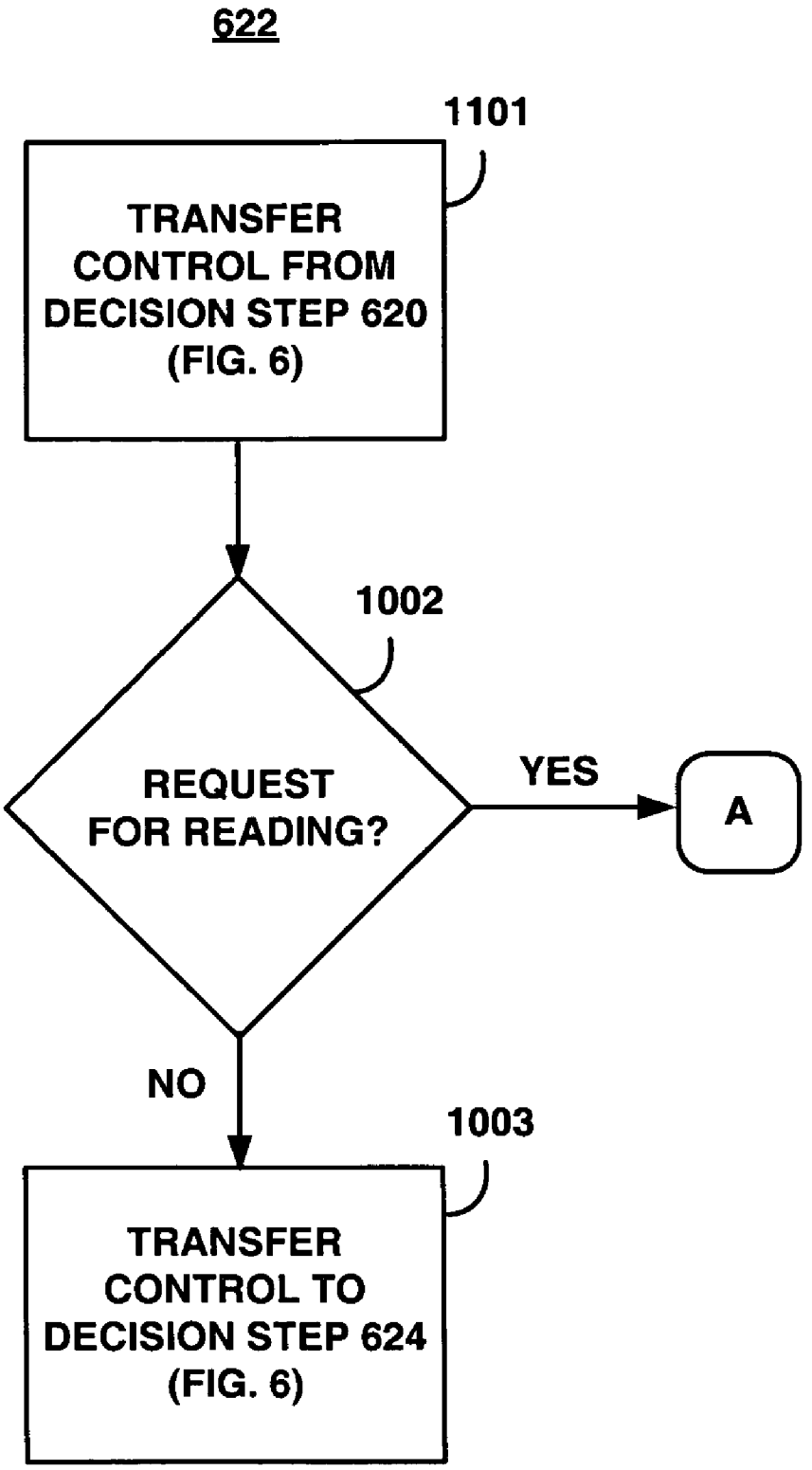

622
1101
TRANSFER CONTROL FROM DECISION STEP 620 (FIG. 6)
1002
REQUEST FOR READING?
YES
A
NO
1003
TRANSFER CONTROL TO DECISION STEP 624 (FIG. 6)

FIG. 11A

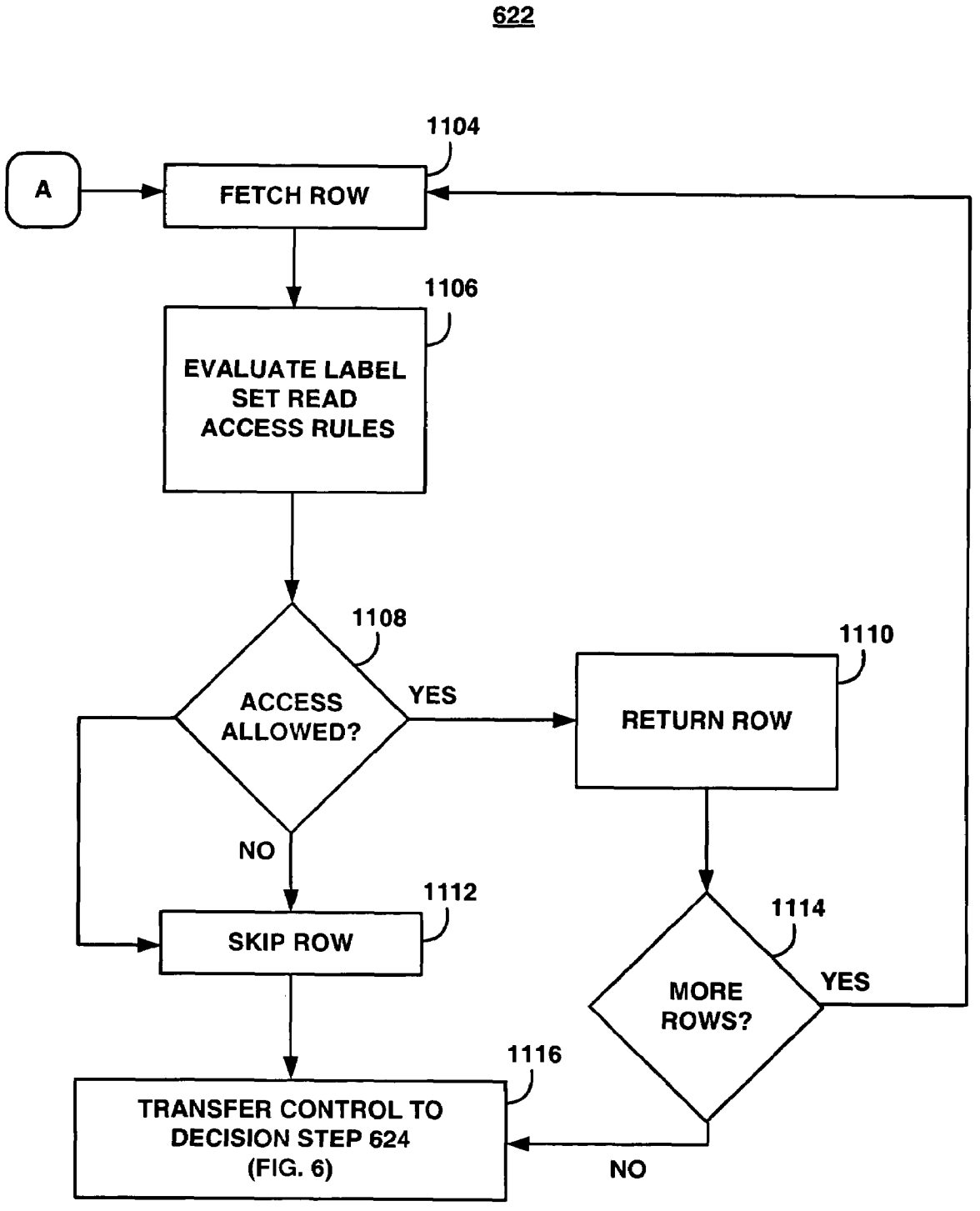

622
A
1104
FETCH ROW
1106
EVALUATE LABEL SET READ ACCESS RULES
1108
ACCESS ALLOWED?
YES
NO
1110
RETURN ROW
1112
SKIP ROW
1114
MORE ROWS?
YES
NO
1116
TRANSFER CONTROL TO DECISION STEP 624 (FIG. 6)

FIG. 11B

CONTROLLING DATA ACCESS USING SECURITY LABEL COMPONENTS

PRIORITY CLAIM

The present application claims the priority of Canadian patent application, Serial No. 2,459,004, titled "Method and System to Control Data Access Using Security Label Components," which was filed on Feb. 20, 2004, and which is incorporated herein by reference.

FIELD OF THE INVENTION

The present invention relates to access control of stored data, and more specifically to a method, a system, and a computer program product to control data access using security label components.

BACKGROUND OF THE INVENTION

In general, access control mechanisms based on labels do not address the requirements from application domains where the label structure and the label access rules do not necessarily match those specific to Multilevel Security (MLS).

Access control regulates the reading, changing, and deleting of objects stored on a computer system. Access control further prevents the accidental or malicious disclosure, modification, or destruction of such objects. Fundamental types of access control comprise discretionary access control (DAC), role-based access control (RBAC), and mandatory access control (MAC). DAC permits the granting and revoking of access privileges to be left to the discretion of the individual users. RBAC does not allow users to have discretionary access to objects. Instead, access permissions are associated with roles; users are made members of appropriate roles. MAC, as defined in the Trusted Computer Security Evaluation Criteria (TCSEC) is "a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity"

One implementation of MAC is Multilevel Security (MLS) that has typically been available primarily on computer and software systems deployed at sensitive government organizations such as the intelligence services or the military.

An MLS model is stated in terms of objects and subjects. An object is a passive entity such as a data file, a record, or a field within a record. A subject is an active process that can request access to objects. The object is assigned a classification, and the subject is assigned a clearance. Classifications and clearances are collectively referred to as access classes or labels. A label is a piece of information that comprises a hierarchical component and a set of unordered compartments.

The hierarchical component specifies the sensitivity of the data. For example, a military organization might define levels top secret, secret, confidential, and unclassified. The compartments component is non-hierarchical and is used to identify areas that describe the sensitivity or category of the labeled data. For example, a military organization might define compartments NATO, nuclear, and army. Labels are partially ordered in a lattice as follows: given two labels L**1** and L**2**, L**1**>=L**2** if and only if the hierarchical component of L**1** is greater than or equal to that of L**2**, and the compartment component of L**1** includes the compartment component of L**2**. L**1** is said to "dominate" L**2**.

MLS restricts data accesses through a simple security property and a *-property (pronounce "the star property"). The simple security property allows a subject read access to an object if and only if the subject's label dominates the object's label. The *-property allows a subject write access to an object if and only if the object's label dominates the subject's label. The *-property prevents subjects from declassifying information.

Even though MLS has traditionally been a requirement of some sensitive government organizations, such as the intelligence services or the military, the ever-increasing customer demand for higher security has made MLS attractive for commercial software products. For example, in certain implementations, the DBMS controls access to database table rows based on a label contained in the row and the label associated with the database user attempting the access. The drawbacks of such implementations comprise a fixed label structure and fixed access rules.

MLS fixes the label structure of a hierarchal component and a set of unordered compartments. Thus, the labels cannot be used for other types of applications to provide fine-grained access control to database table rows. For example, in certain banking applications, a label represents a geographical location, which is a single component and is not hierarchal. MLS further fixes access rules. Access to database table rows is governed by the simple security property and the *-property. Thus, this form of access control based on labels cannot be used for other purposes. For example, banking applications have different requirements for the label structure and for the label access rules.

Although this technology has proven to be useful, it would be desirable to present additional improvements. Existing access control systems based on labels strictly implement the MLS semantics. These conventional access control systems fail to address the label requirements from application domains where the label structure and the label access rules do not necessarily match those described in MLS. Moreover, these existing solutions cannot be used to enforce privacy policies. Generally, a privacy policy indicates for which purposes an information is collected, whether or not the collected information will be communicated to others, and for how long the collected information is retained before it is discarded.

For example, a user should not be able to access a customer record for the purpose of sending that customer marketing information if that customer did not agree to receipt of such information. Access to privacy-sensitive data can be regarded as analogous to access to labeled data. In both cases, a tag is associated with the object being accessed and the subject accessing that object. The tag is a "purpose" in the case of the accessing privacy-sensitive data and a "label" in the case of the accessing labeled data.

However, existing access control solutions based on labels strictly implement the MLS semantics, and thus cannot be used to enforce privacy policies for the following reasons. Labels include a hierarchal component that is not applicable in the case of privacy. Furthermore, the MLS security properties do not apply in the context of privacy.

What is therefore needed is a system, a computer program product, and an associated method for a label-based access control (LBAC) solution that is capable of implementing the MLS semantics and of addressing the requirements from a

variety of application domains, including MLS requirements. The need for such a solution has heretofore remained unsatisfied.

SUMMARY OF THE INVENTION

The present invention satisfies this need, and presents a system, a service, a computer program product, and an associated method (collectively referred to herein as "the system" or "the present system") for controlling data access using security label components. The present system provides, for a data processing system having memory for storing data elements, a method for directing the data processing system to control user access to the stored data elements.

Each stored data element is associated with a set of data security label components. Each user is associated with a set of user security label components. The present system comprises receiving a user request to access the stored data elements, comparing the set of user security label components against the set of data security label components associated with the users, and determining whether to permit access to the stored data responsive to the received user request based on results of the comparison.

The present system comprises a computer program product for directing a data processing system to control user access to data elements stored in memory of the data processing system. Each stored data element is associated with a set of data security label components. Each user is associated with a set of user security label components. The computer program product comprises a computer readable transport medium for transporting computer executable code to the data processing system. The computer executable code comprises computer executable code for receiving a user request to access the stored data elements, computer executable code for comparing the set of user security label components against the set of data security label components associated with the users, and computer executable code for determining whether to permit access to the stored data responsive to the received user request based on results of the comparison.

The present system comprises an access control system to be operatively coupled to a data processing system having memory for storing data elements. The access control system directs the data processing system to control user access to the stored data elements. Each stored data element is associated with a set of data security label components. Each user is associated with a set of user security label components. The access control system comprises means for receiving a user request to access the stored data elements, means for comparing the set of user security label components against the set of data security label components associated with the users, and means for determining whether to permit access to the stored data responsive to the received user request based on results of the comparison.

BRIEF DESCRIPTION OF THE DRAWINGS

The various features of the present invention and the manner of attaining them will be described in greater detail with reference to the following description, claims, and drawings, wherein reference numerals are reused, where appropriate, to indicate a correspondence between the referenced items, and wherein:

FIG. **1** is a schematic illustration of an exemplary database management system installed on a data processing system having memory storing a database in which an access control system (ACS) of the present invention can be used;

FIG. **2** is a table illustrating types of access parameters implemented by the access control system of FIG. **1**;

FIG. **3** is a table illustrating data and table access parameters of the access control system of FIG. **1** for the database of FIG. **1**;

FIG. **4** is a table illustrating a user access table in which user access parameters are associated by the access control system of FIG. **1** with users of the database of FIG. **1**;

FIG. **5** is a table illustrating tests used by the access control system of FIG. **1** in comparing table access parameters against user access parameters for access to the database of FIG. **1**;

FIG. **6** is a process flow chart illustrating a method of operation of the access control system of FIG. **1**, in which the operation comprises determining user requirements;

FIG. **7** is a process flow chart illustrating a method of operation of the access control system of FIG. **1**, in which the operation comprises defining access parameter types and associated tests;

FIG. **8** is a process flow chart illustrating a method of operation of the access control system of FIG. **1**, in which the operation comprises creating a table contained in the database of FIG. **1**;

FIG. **9** is a process flow chart illustrating a method of operation of the access control system of FIG. **1**, in which the operation comprises assigning user access parameters;

FIG. **10** is a process flow chart illustrating a method of operation of the access control system of FIG. **1**, in which the operation comprises writing data to a table contained in the database of FIG. **1**; and

FIG. **11** is a process flow chart illustrating a method of operation of the access control system of FIG. **1**, in which the operation comprises reading data from a table contained in the database of FIG. **1**.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The following detailed description of the embodiments of the present invention does not limit the implementation of the embodiments to any particular computer programming language. The computer program product may be implemented in any computer programming language provided that the OS (Operating System) provides the facilities that may support the requirements of the computer program product. A preferred embodiment is implemented in the C or C++ computer programming language (or may be implemented in other computer programming languages in conjunction with C/C++). Any limitations presented would be a result of a particular type of operating system, computer programming language, or data processing system and would not be a limitation of the embodiments described herein.

FIG. **1** portrays an exemplary overall environment in which a system and associated method for controlling data access using security label components (an access control system **115**) according to the present invention may be used. The access control system **115** comprises a software programming code or a computer program product that is typically embedded within, or installed on a memory **112**. Alternatively, system **10** can be saved on a suitable storage medium such as a diskette, a CD, a hard drive, or like devices.

A data processing system (DPS) **100** comprises a Central Processing Unit (CPU) **102** operatively coupled to a bus **104**. Bus **104** is operatively coupled to I/O (Input/Output Interface Unit) **105** and coupled to memory **112**. I/O **105** operatively couples bus **104** to a display unit **108**, a keyboard/mouse (keyboard **110**), a disc **111**, and a network **109**. Memory **112**

may comprises a combination of many types of memory, such as RAM (Random Access Memory), ROM (Read Only Memory), and hard disk (not illustrated).

The memory **112** stores a database **116** and a database management system (DBMS) **114**. The DBMS **114** comprises the access control system **115**. However, the access control system **115** may operate independently of the DBMS **114** and there may be system calls transferred between the DBMS **114** and the access control system **115**. The DBMS **114** and the access control system **115** comprise computer executable code that is executed by the CPU **102**. The computer executable code is compiled from computer programmed instructions written in a high-level computer programming language (such as, for example, C++ or Java). The computer executable code is loaded to memory **112** by transferring the computer executable code from disc **111**.

Disc **111** is a computer program product comprising a computer readable medium that is used to transport the computer executable code to the DPS **100** via I/O **105**. Alternatively, the computer readable medium comprises a computer readable transport signal carried by network **109**, the signal being used to transport the computer executable code to the DPS **100** via I/O **105**. It will be appreciated that the computer executable code configures the DPS **100** (which is a general purpose machine) into a specifically configured machine that may be treated as comprising modules or mechanisms that achieve specific functions (these functions to be described below in more detail).

Generally, the computer executable code included in the access control system **115** directs CPU **102** to define security labels for data and users. Data security label components are found in types of access parameters **118**. The data security label components are associated with each data element stored in a classified table **120**. The access control system **115** also defines user security label components that are stored in a user access table **122**. Each user security label component is associated with a user. The access control system **115** directs CPU **102** to determine whether the user, who submitted a request to access a data element, is granted access or is denied access to the data element based upon a comparison made between the user security label components and the data security label components. Tests **124** comprise these tests or rules for allowing user access to the data element.

The access control system **115** is used to control user access to stored data shown in classified table **120**. Associated with the stored data are security label components. Associated with the users are user security label components. The access control system **115** configures a configurable security label structure that describes the security label components associated with the stored data and the users (the security label structure is described below in greater detail). The access control system **115** also defines label access rules to be associated with the configurable security label structure. The access control system **115** executes the defined label access rules to compare the security label components associated with the stored data against the security label components associated with the users. The access control system **115** determines whether to permit and to not permit user access to the stored data based on the outcome of the executed defined label access rules.

FIG. **2** is a table illustrating the types of access parameters used by the access control system **115** of FIG. **1**, access parameters **118**. These types of access parameters are a collection of security access parameters further referenced herein as a security label set label set **118** or label set **118**. The label set **118** is a security label structure that comprises types of security access components (label components) **202**, each associated with security access parameters such as label component names **204**.

The label components label components **202** is a set of security access label components that are organized as a schema; the schema is the label set **118**. As a table schema defines the set of columns that make up a data row, so the label set **118** represents a schema that defines a set of label components **202** that make up a security access label. The security access label is either associated with a data element stored in classified table **120** or associated with a user—as indicated in user access table **122**. The label set **118** comprises security access rules that the access control system **115** uses to determine whether a user who is associated with a label, L_**1**, may be granted or may be denied access to a data element associated with a label, L_**2**. Further description for the access rules or tests is provided below. The security access rules (or tests) may be stored in a test table, tests **124** of FIG. **1**.

A type of access parameter may be treated as one of the label components **202**, each of which is associated with one of the label component names **204**. The set of label components **202** is an entity that may be created, dropped, and altered by the access control system **115**. The security label set **118** (to be associated with a data element or with a user) may include one or more of the label components **202**. There may be types of the label components **202**, such as for example a "set" type of the label components **202** and a "tree" type of the label components **202**. There may be an ordered set type of the label components **202** and there may be an unordered set type of the label components **202**.

In an ordered set type of the label components **202**, the order in which element in a component appears is important: for example, the rank of a first element is higher than a rank of a second element, a rank of a second element is higher than a rank of a third element and so on (for one of the label components **202**). An example of the types of components is indicated in row **206** of the label set **118**, examples of elements of the label component **202**.

A tree type of the label components **202** represents a hierarchy of an organization (such as a company for example). The tree type of the label components **202** may be used to represent organizational charts and/or to identify departments within an organization that owns the data stored in the classified table **120**. The label components **202** are stored in the label set **118**, for example, or stored in a database system catalog if the access control system **115** is to be implemented in DBMS **114**.

FIG. **3** shows the classified table **120** of FIG. **1**. A classified table is a database table that comprises labeled data rows. When a database administrator marks the classified table **120** as classified, the database administrator specifies the label set **118** to be used or associated with the classified table **120**. The label set **118** determines the structure of the label components **202** to be used to label the data rows of the classified table **120**. The label set **118** further determines the label access rules (tests **124**) to be used for enforcing access to the classified table **120**.

The classified table **120** comprises one or more classified data elements **303**. The classified table **120** further comprises one or more row labels **306**, one or more row labels **308**, and one or more row labels **310**. Each of the row labels **306**, row labels **308**, and row labels **310** are associated with a data element PLAN_A, PLAN_B, PLAN_C, and PLAN_D, respectively, and are indicated in respective table row **312**, table row **314**, table row **316**, and table row **318**. The access control system **115** generates and assigns security access labels; i.e., access labels and row labels

A row label is assigned to each data element stored in the classified table **120**. The data element may be a picture, a test document, or combination thereof. It is understood that each row has its own row label (there cannot be duplicate row labels). It is possible that two rows in the classified table **120** may have two row labels that are identical.

The classified table **120** is a convenient organized storage of a plurality of data elements used to illustrate one embodiment. The row label contains components that are used to express or indicate the access requirements of a data element. For example, row label of PLAN_A (see table row **312**) comprises security label components LEVEL=TOP SECRET, COMPARTMENT=ARMY, OWNER=MARINES.

For example, for PLAN_A of table row **312**, if a user is a member of MARINES division of ARMY and that user has a classification clearance of at least TOP SECRET or better, that user may have read and/or write access to PLAN_A. However, if that user is not a member of MARINES division but is instead a member of any other division of ARMY and that user also has a classification level of at least TOP SECRET or better, then that user may have only read access to PLAN_A. For any other condition, that user may not have read or write access to PLAN_A.

For example, for PLAN_B of table row **314**, if a user is a member of RESEARCH division of NASA and that user also has a classification clearance of at least SECRET or better, that user may have read and write access to PLAN_B. However, if that user is not a member of RESEARCH division of NASA but that user is a member of some other NASA division and that user has a classification level of at least SECRET or better, that user may have only read access to PLAN_B. For any other condition, that user may not have read or write access to PLAN_B.

DBMS **114** may comprise a function that allows database users to refer to the security label associated with a row in a classified table in SQL statements. This function may, for example, be called "ROWLABEL". ROWLABEL can be referenced in an SQL statement. ROWLABEL allows users to reference a row label in SQL statements for manipulating data contained in the rows of the classified table **120**.

For SELECT statements and WHERE clauses (to be included in an SQL statement), individual label components are referenced by providing the component name as a parameter to the ROWLABEL function. For example, a user who wishes to select only the level component of a label can issue the following SQL statement:

SELECT ROWLABEL(level), . . . , FROM T**1**

If the user wishes to express a predicate, the following SQL statement can be issued:

SELECT ROWLABEL(level), . . . , FROM T**1** WHERE ROWLABEL(level)='Secret'

For INSERT and UPDATE SQL statements, ROWLABEL is a means of providing the label value of a data row. For example, a user who wishes to insert a row into a classified table can issue the following SQL statement:

INSERT INTO T**1** VALUES (ROWLABEL('SECRET', 'NATO'), . . .)

A user who wishes to update the level component in the label of some data row can issue the following SQL statement:

UPDATE T**1** SET ROWLABEL(level)=ROWLABEL ('SECRET') WHERE C**1**=5

FIG. **4** shows the user access table **122** of FIG. **1**. The user access table **122** comprises security access labels (having component **406**, component **408**, and component **410**) associated with user identifiers (column **402**). An access label is assigned to each user. It is possible that users may have identical access labels. Access labels may be granted and revoked by the database administrator (that is, an executive level user of the access control system **115**) or by another database user who has sufficient authority to act as an administrator. Access labels may be stored, for example, in a database catalog. The access label comprises components that express or indicate user ability to access data elements stored in the classified table **120** as predetermined by the administrator.

For example, user WALID (row **412**) has a LEVEL=TOP SECRET (that is, Walid has top secret classification clearance). For WALID, COMPARTMENT=ARMY and NASA (that is, user Walid is a member of the ARMY and a member of NASA). Also, user Walid is indicated as an owner of documents that belong to the MARINES (a division of ARMY). These values indicate that user Walid may have only read and/or write access to data elements associated with a security label component MARINES provided that user Walid has the proper security clearance level (in this case, the security clearance of user Walid is TOP SECRET). Furthermore, user Walid may have only read access to any data element associated with a security label component ARMY or NASA, provided that user Walid has the proper security clearance level (in this case, the security clearance of user Walid is TOP SECRET).

For example, if a data element is associated with a clearance LEVEL that is greater than TOP SECRET (and associated with ARMY or NASA), user Walid may not have read access to that data element because the classification LEVEL of user Walid is not sufficient.

User BIRD (row **414**) may have read and/or write access to any data elements that are associated with RESEARCH division of NASA provided the LEVEL classification of user Bird is sufficient to permit user BIRD access to those data element.

User BIRD may have only read access to data elements associated with NASA that do not belong with the RESEARCH division of NASA (provided that the LEVEL classification of user Bird is sufficient to permit user BIRD read access to those sorts of data elements).

FIG. **5** shows the tests **124** of FIG. **1**. The tests **124** are to be selected and the label set **118** may also specify the access rules or tests that the access control system **115** uses to determine whether a user who is associated with an access label (i.e., access label_**1**) may have access to a data element associated with a row label (i.e., row label_**1**).

Label access rules may be divided categories such as read access rules and write access rules. The read access rules are used by the access control system **115** when a user attempts to read a data element from the classified table **120** (for example, when the user submits a SELECT statement to the DBMS **114**). The access control system **115** uses the write access rules when a user attempts to write (such as, performing an insert, an update or a delete command) a data element. A label access rule may be a predicate that combines the same label components contained in an access label and a row label by using an operator as follows (for example):

Access Label Component_A <operator> Row Label Component_A

The type of operator to be used in the label access rules may depend on the type of label component. For ordered sets of label components, the operator may be any of the following relational operators {=, <=, <, >, >=, !=}. For non-ordered sets of label components, the operator may be, for example, any one of the set operators {IN, INTERSECT. For trees of label components, the operator may be, for example, the INTERSECT set operator. The label set **118** and label access rules

may be stored in a database system catalogs when the access control system **115** is integrated with the DBMS **114**.

Exceptions to the label access rules here provide a flexibility to bypass one or more label access rules. For example, in an MLS context, it is often the case that some special users are allowed to write information to data elements associated with lower security levels even though this is in contradiction with the *-security property. Thus, exceptions are introduced to allow the database administrator to grant a database user an exception to bypass one or more rules associated with a particular label set.

FIG. **6** illustrates a method **600** of operation of the access control system **115** of FIG. **1**, in which the method **600** comprises determining user commands and requirements. The access control system **115** of FIG. **1** begins operation at step **602**.

The access control system **115** determines whether the user desires to create the label set **118** of FIG. **2** or create the tests **124** of FIG. **5** (decision step **604**). If the user desires to create label set **118** or tests **124**, the access control system **115** creates access parameter types and tests (step **606**). If the user does not desire to create label set **118** or tests **124**, operation continues to decision step **608**.

The access control system **115** determines whether the user desires to create the classified table **120** of FIG. **3** (decision step **608**). If the user desires to create the classified table **120**, the access control system **115** creates the classified table **120** (step **610**). If the user does not desire to create the classified table **120**, operation continues to decision step **612**.

The access control system **115** determines whether the user desires to assign security access labels to users as shown in user access table **122** of FIG. **4** (decision step **612**). If the user desires to assign security access labels, the access control system **115** assigns user access parameters (step **614**). If the user does not desire to assign security access labels, operation continues to decision step **616**.

The access control system **115** determines whether the user desires to write data to classified table **120** of FIG. **3** (decision step **616**). If the user desires to write data to classified table **120**, the access control system **115** writes data to the classified table **120** (step **618**). If the user does not desire to write data to classified table **120**, operation continues to decision step **620**.

The access control system **115** determines whether the user desires read data (that is, data elements **303**) from the classified table **120** of FIG. **3** (decision step **620**). If the user desires to read data from the classified table **120**, the access control system reads data from the classified table (step **622**). If the user does not desire to read data from the classified table **120**, operation continues to decision step **624**.

The access control system **115** determines whether the user desires to re-perform any of operations of decision step **604**, decision step **608**, decision step **612**, decision step **616**, or decision step **620** (decision step **624**). If the user desires to re-perform any of these operations, the access control system **115** returns to decision step **604** and repeats steps **604** through **622** as required. If the user does not desire to perform these operations, access control system **115** halts any further operations (step **626**).

FIG. **7** illustrates a method of operation of step **606** of the method **600** of the access control system **115** of FIG. **1**. Step **606** comprises defining the label set **118** of FIG. **2**. The label set **118** is a set of types of access components. Step **606** further comprises defining the label access rules (tests **124** of FIG. **5**) to be associated with the label set **118**.

The access control system **115** helps the database administrator (an executive user of the access control system **115**) to define the security label components (indicated in row **202** of label set **118**) and their types. For example, the access control system **115** permits the database administrator to define security a label component referenced as LEVEL (of type integer) and a label component referenced as COMPARTMENT (of type string).

The access control system **115** permits the database administrator to define the label set that comprises the security label component **202**. The relationship between the security label component **202** and the label set **118** is analogous to the relationship between a data row of a table and a table schema. As the table schema defines the set of columns that make up a data row, so the label set **118** set defines the set of security label components that make up the label set **118**. The label set **118** may also be associated with a test table, tests **124** of FIG. **5**. The test table, tests **124**, comprises a set of access rules that the access control system **115** uses to determine whether a user who is associated with a security access label, L_**1**, may or may not access a data row associated with a security label, L_**2**. The label access rules may be divided into categories such as read access rules and write access rules.

The access control system **115** transfers control from decision step **606** of FIG. **6** because a user has indicated a desire to define the components to be included in the label set **118** of FIG. **2** and the tests **124** of FIG. **5** (step **702**).

The access control system **115** defines the components of label set **118** of FIG. **2** (step **704**). The components **202** of label set **118** indicate the types of access parameters **306**, **208**, **310** to be associated with data elements **303** of FIG. **3**.

The access control system **115** defines the tests **124** of FIG. **5** to be associated with the components **202** of label set **118** (step **706**). The access control system transfers control back to decision step **608** of FIG. **6**.

FIG. **8** illustrates a method of operation of step **610** of the method **600** of the access control system **115** of FIG. **1**. Step **610** comprises creating the classified table **120** of FIG. **1**.

A database administrator (an executive user of the access control system **115**) attaches the label set **118** to the classified table **120**. When the label set **118** is attached to the classified table **120**, the table **120** is considered classified; i.e., the data elements may only be accessed depending on the execution outcome of the tests **124** of FIG. **5**.

When the user desires to access data elements contained in the classified table **120**, the access control system **115** applies the access rules defined and associated with the label set **118** of FIG. **2**. The label set **118** is attached to the classified table **120** to determine whether or not a user may have or may not have access to a row containing a data element within the classified table **120**.

The access control system **115** transfers control from decision step **610** of FIG. **6** because a user has indicated a desire to create the classified table **120** of FIG. **3** (step **802**). The access control system **115** sets up the classified table **120** (step **804**).

The access control system **115** generates a column **302** to contain the data element identifiers (step **806**). Each of these identifiers identifies a specific data element contained in table **120**. The access control system **115** generates a column **303** to contain the data elements (step **808**).

The access control system **115** generates a column for each row label component **306**, **308** and **310** (that is, each user Access Label component **306**,**108**, **310**) (step **810**). Each component **306**, **308**, **310** indicates the data element access requirements to be compared against user access label components at a later time (the comparison is further described below). The access control system transfers control back to decision step **612** of FIG. **6** (step **812**).

FIG. **9** illustrates a method of operation of step **614** of the method **600** of the access control system **115** of FIG. **1**. Step **614** comprises assigning user access labels to users. Each access label (security Access Label) comprises user access components, each component indicating an ability of a user to access data elements stored in the classified table **120** of FIG. **3**.

The access control system **115** permits a database administrator (who is an executive level user of the access control system **115**) to grant access labels (security Access Labels) to specific database users. The access control system **115** uses the access labels in conjunction with the label set access rules to determine user access rights with respect to rows (that is, data elements associated with a row) contained in the classified table **120**. The access control system **115** may permit the database administrator to choose to grant one or more exceptions to a database user to allow them to bypass one or more access rules associated with the label set **118**.

The access control system **115** may be integrated into an SQL (Structured Query Language) compiler component (not illustrated) of the DBMS **114** such that when an SQL query references the classified table **120**, the SQL compiler incorporates the access rules of the label set associated with the classified table **120** in an access plan. The SQL compiler generates the access plan). The access plan is used to execute the compiled user SQL query. When the access plan is executed, the access rules may be evaluated for each row (that contains the data element) in the classified table **120** to determine whether access to a specific row should be allowed or disallowed.

The access control system **115** transfers control from decision step **614** of FIG. **6** because a database administrator indicated a desire to assign user access parameters to a user (step **902**).

The access control system **115** determines whether the user request is a request to generate the user access table **122** (decision step **904**). If the user request indicates a desire to generate the user access table **122**, the user access table **122** is generated (step **906**) and processing continues to decision step **908**. If the user request indicates no desire to generate the user access table **122**, processing continues to operation decision step **908**.

The access control system **115** determines whether the received user request indicates a desire to assign access labels (security Access Labels) to a specific user (decision step **908**). If it is determined that the user wishes to assign an access label to the specific user, the access control system assigns an access label to a user (step **910**) and components of the access label are selected or filled in for the access label assigned to the specific user (step **912**). If it is determined that the user does not wish to assign an access label to the specific user, the access control system **115** transfers control to decision step **616** of FIG. **6**.

FIG. **10** illustrates a method of operation of step **618** of the method **600** of the access control system **115** of FIG. **1**. Step **618** comprises writing data elements to the classified table **120** of FIG. **1**. The access control system transfers control from decision step **618** of FIG. **6** step **1001**).

The access control system **115** determines whether the access control system **115** received a user request for writing (that is, a write access command) data to a data element stored in the classified table **120** (decision step **1002**). If the user request is not a write request, the access control system returns to step **616** of FIG. **6**. If the user request indicates a write access request, the access control system **115** proceeds to step **1004**. The access control system **115** receives a row to be written (step **1004**).

The access control system **115** validates row security label components associated with the row (that is, the data element) to be written to the classified table **120** (decision step **1006**). If the row security label components are not valid the access control system returns an error to the user (step **1012**) and then transfers control to decision step **620** of FIG. **6** (step **1018**). The row security label components are not valid if the row security label components are not composed of the exact same components defined in the label set associated with the classified table **120** or if the values of each row security label component are not valid with respect to their type.

If the row security label components are valid (decision step **1006**), the access control system evaluates write access rules associated with the label set of the classified table **120** (step **1008**).

The access control system **115** determines whether the access may be allowed (decision step **1010**). If it is determined that access may be allowed, the access control system **115** writes the row into the classified table **120** (step **1014**). If it is determined that access may not be granted or not be allowed, the access control system **115** returns an error indication to the user (step **1012**) and returns to decision step **620** of FIG. **6** (step **1018**).

The access control system **115** determines whether there are more rows to process (decision step **1016**). If it is determined that more rows are to be processed, the access control system returns to step **1004** and repeats step **1004** through step **1010** for the next row received. If it is determined that there are no more rows to be written to the classified table **120**, the access control system returns to decision step **620** of FIG. **6** (step **1018**).

FIG. **11** illustrates a method of operation of step **622** of method **600** of the access control system **115** of FIG. **1**. Step **622** comprises reading one or more rows that were written into the classified table **120** of FIG. **1**. The access control system **115** transfers control from decision step **622** of FIG. **6** (step **1101**)

The access control system **115** determines the type of access request requested by a user (decision step **1102**). If the type of user access being requested is a read access, the access control system **115** proceeds to step **1104**. If the type of user access being requested is not a read access, operation is transferred to decision step **624** of FIG. **6**.

The access control system **115** fetches the next row in the classified table **120** (step **1104**). The access control system **115** evaluates the read access rules associated with the label set **118** (step **1106**).

The access control system **115** determines whether user access may be granted or allowed (decision step **1108**). If the determination is made that user access may be allowed, the access control system **115** returns the fetched row to the user (step **110**). If the determination is made that the user may not be allowed or may not be granted access, the access control system **115** skips the fetched row (i.e., the fetched row is not returned to the user) (step **1112**).

The access control system **115** determines whether there are any more rows in the classified table **120** to be fetched. If there are no more rows to be fetched, the access control system **115** returns to decision step **624** of FIG. **6**. If there are more rows to be fetched, the access control system **115** returns to step **1104** in which case the next row in the classified table **120** is fetched and step **1104** to step **1114** may be repeated as needed.

In one embodiment, the access control system **115** uses security access labels to provide fine-grained access control in the DBMS of FIG. **1**. Generally, fine-grained access control refers to a method of providing row-level security for a table

as known to those skilled in the art. In private banking, country laws and regulations often require limitation of the amount of data that can be viewed by a bank employee. For example, Swiss banking laws do not allow a Swiss bank employee located in Toronto to access account information for customers based in Switzerland. A bank employee can only access account information for customers who are based in the same location as the bank employee.

Typically, the bank addresses this access control problem as follows. When a bank employee is authenticated, a security context is assigned to him/her based on the authentication type, location, geography, etc. When that bank employee issues a request, the request goes through a number of systems up to a mainframe system where an application picks it up and adds an appropriate predicate based on the employee location (e.g., WHERE location="Toronto") before it is submitted to the DBMS. This solution is error prone and exposes security policies directly to the application programmers. It also requires many code reviews to ensure correctness.

The problem stated above can be easily solved using the control access system **115** by associating a label with each customer account that specifies its location and by associating a label with each bank employee that specifies where that employee is located. The DBMS can then ensure that bank employees can only access account information for the customers located in their geographical location.

Referring to FIG. **7**, the following SQL statement creates a label component called location:

CREATE LABEL COMPONENT location OF TYPE varchar(**15**)

USING SET ("Zurich", "Toronto", "London", "Paris")

The following SQL statement creates a label set based on the component defined above:

CREATE LABEL SET set**1** COMPONENTS location

READ ACCESS RULE rule**1** ACCESS LABEL location IN ROW LABEL location

WRITE ACCESS RULE rule**2** ROW LABEL location IN ACCESS LABEL location

Referring to FIG. **8**, the following SQL statement creates a classified table T**1** to store customer account information and associates this table with label set set**1**:

CREATE Table T**1** (CustomerID int, CustomerName char (**30**), CustomerBalance)

LABEL SET set**1**

Referring to FIG. **9**, the following SQL statements create two access labels and grant them to bank employee empA and empB:

CREATE ACCESS LABEL label**1** IN LABEL SET set**1** Location "Toronto"

CREATE ACCESS LABEL label**2** IN LABEL SET set**1** Location "Zurich"

GRANT LABEL label**1** FOR USER empA FOR ALL

GRANT LABEL label**2** FOR USER empB FOR ALL

Referring to FIG. **10**, when a user issues an SQL statement against the classified table T**1** that reads or modifies a data row, the label access rules defined above are evaluated to determine whether or not the user can read/modify the data row. Below are exemplary INSERT SQL statement examples for user empA.

SQL Command	Status
INSERT INTO T1 VALUES (1, 'Hans',100, ROWLABEL ('Zurich'))	This command is rejected because user empA is not allowed to write account information for customers located in Zurich (rule2).

-continued

SQL Command	Status
INSERT INTO T1 VALUES (2, 'PBIRD',100,ROWLABEL ('Toronto'))	This command is accepted because rule2 is satisfied.
INSERT INTO T1 VALUES (3,'WRJAIBI',10,ROWLABEL ('Toronto')	This command is accepted because rule2 is satisfied.

Below are exemplary INSERT SQL statement examples for user empB:

SQL Command	Status
INSERT INTO T1 VALUES (1,'Hans',100, ROWLABEL ('Zurich'))	This command is accepted because rule2 is satisfied.
INSERT INTO T1 VALUES (4,'Urs',100, ROWLABEL ('Zurich'))	This command is accepted because rule2 is satisfied.

Referring to FIG. **11**, the following are exemplary SELECT SQL statement examples for user empA.

SQL Command	Status
SELECT * FROM T1	This command returns only rows PBIRD and WRJAIBI. The other 2 rows are not returned because rule 1 is not satisfied.

The following are exemplary SELECT SQL statement examples for user empB.

SQL Command	Status
SELECT * FROM T1	This command returns only rows Hans and Urs. The other 2 rows are not returned because rule 1 is not satisfied.

In the example described above, Urs is a first name commonly used in the German part of Switzerland. In this case, the access control system **115** is inserting a record for the customer called Urs.

In a further example, a bank executive (exec**1**) located in Zurich holds access label label**1** and is permitted read access to account information for customers located in Toronto. The administrator can grant a label exception to this executive to bypass rule **1** as follows:

GRANT LABEL EXCEPTION ON RULE rule**1** IN set**1** TO USER exec**1**

If the executive issues the SELECT * FROM T**1** query, he/she will be able to see all the rows above.

In a further embodiment, the access control system **115** uses security access labels for providing MLS capability in the DBMS **114** of FIG. **1**. An application wishes the DBMS **114** to provide MLS semantics. In MLS, a label comprises two components: a hierarchical component a set of unordered compartments. The hierarchical component is referenced as a level. In an example, the valid values a level comprises are

Top Secret, Secret, Classified, and Unclassified. Similarly, a compartment can take any of the following values: NATO, Nuclear and Army.

Referring to FIG. **7**, the following two SQL statements can be used to create the two components.

CREATE LABEL COMPONENT level OF TYPE varchar (**15**)
USING ORDERED SET ("TOP SECRET", "SECRET", "CLASSIFIED", "UNCLASSIFIED")
CREATE LABEL COMPONENT compartments OF TYPE varchar(**15**)
USING SET ("NATO", "Nuclear", "Army")

The keyword ORDERED in the definition of the first component indicates that the order in which the elements appear in the set is significant.

Referring to FIG. **7**, the access control system **115** uses the following SQL statement to create a label set **118** where each label is composed of the two components defined above. The statement also permits the access control system **115** to specify the label access rules. These label access rules implement the simple security property and the *-property previously described.

CREATE LABEL SET set**1** COMPONENTS level, compartments
READ ACCESS RULE rule**1** ACCESS LABEL level>=ROW LABEL level
READ ACCESS RULE rule**2** ROW LABEL compartments IN ACCESS LABEL compartments
WRITE ACCESS RULE rule**3** ROW LABEL level>=ACCESS LABEL level
WRITE ACCESS RULE rule**4** ACCESS LABEL compartments IN ROW LABEL compartments

Referring to FIG. **8**, the application wishes to create a table where each data row is to be labeled using a label from set**1** above. The access control system can use the following SQL statement can be used to generate such a table.

CREATE Table T**1** (C**1** char(**3**), C**2** int)
LABEL SET set**1**

Referring to FIG. **9**, the access control system **115** generates the access labels and assigns the access labels to database users using the following SQL statements:

CREATE ACCESS LABEL label**1** IN LABEL SET set**1**
Level "TOP SECRET", compartments "Nuclear"
CREATE ACCESS LABEL label**2** IN LABEL SET set**1**
Level "CLASSIFIED", compartments "Army"
GRANT LABEL label**1** FOR USER walid FOR ALL
GRANT LABEL label**2** FOR USER paul FOR ALL

Referring to FIG. **10**, when a user issues an SQL statement against the classified table T**1** that reads or modifies a data row, the label access rules defined above are evaluated to determine whether or not the user can read/modify the data row. Below are exemplary INSERT SQL statements for user walid.

SQL Command	Status
INSERT INTO T1 VALUES ('abc',1,ROWLABEL('TOP SECRET', 'NATO'))	This command is rejected because the compartment of user walid (Nuclear) is not included in the compartments of the row being inserted (rule4).
INSERT INTO T1 VALUES ('def',2,ROWLABEL('TOP SECRET', 'Nuclear'))	This command is accepted because both rule3 and rule4 are satisfied.
INSERT INTO T1 VALUES ('ghi',3,ROWLABEL ('UNCLASSIFIED','Nuclear'))	This command is rejected because user walid is attempting to write a row at a lower security level (level 3).

Below are exemplary INSERT SQL statements for user paul.

SQL Command	Status
INSERT INTO T1 VALUES ('jkl',4,ROWLABEL ('CLASSIFIED','Army'))	This command is accepted because both rule3 and rule4 are satisfied.
INSERT INTO T1 VALUES ('mno',5,ROWLABEL ('SECRET','Army'))	This command is accepted because both rule3 and rule4 are satisfied

Referring to FIG. **11**, the following are exemplary SELECT SQL statements for user walid.

SQL Command	Status
SELECT * FROM T1	This command returns only row: ('def',2,{'TOP SECRET', 'Nuclear'}). The other 2 rows are not returned because rule 2 is not satisfied.

The following are exemplary SELECT SQL statements for user paul.

SQL Command	Status
SELECT * FROM T1	This command returns only row: ('jkl',4,{'CLASSIFIED', 'Army'}). The other 2 rows are not returned because rule 1 is not satisfied.

The access control system **115** may be included in a database management system (DBMS) **114** or information retrieval system (IRS). Further, the access control system may be included in many types of software applications, such as, for example (the following represents a non-exhaustive list of such applications):

a DBMS adapted to provide fine-grained access control to database table rows;
a DBMS adapted to provide MLS;
a DBMS adapted to enforce privacy policies;
an operating system (OS) stored in the memory of a DPS, the OS being adapted to implement a policy where access to systems files is based on security labels and label access rules;
a Publish/Subscribe system adapted to implement a policy where the matching process also take into account the security labels associated with a subscription and an event as well as the label access rules; and
an XML system adapted to control access to the nodes in an XML document based on the security labels and label access rules.

The access control system **115** is an improvement over known LBAC solutions in the sense that the access control system **115** is not restricted to MLS semantics. The access control system **115** may be used in various application domains and for various purposes. The access control system **115** may also be used to provide.

It is to be understood that while specific embodiments have been described to illustrate certain applications of the principle of the present invention. Other modifications are possible without departing from the spirit and scope of the present invention.

What is claimed is:

1. A computer-implemented method of controlling user access to stored data elements, comprising configuring one or more computer processors to perform an operation comprising:

defining, based on user input, an ordered plurality of security levels that describe sensitivity of the stored data elements;

defining, based on user input, a plurality of categories that categorize the stored data elements;

associating each user with a security level from the ordered plurality of security levels and a category from the plurality of categories, thereby defining a security clearance for each respective user; and

defining, based on user input and for each of the stored data elements, a read access rule for the respective stored data element, wherein the read access rule comprises a condition for granting read access to the respective stored data element, the condition specifying a security level of the plurality of security levels and a category from the plurality of categories.

2. The computer-implemented method of claim 1, wherein the operation further comprises:

receiving a user request to access a stored data element;

evaluating, by operation of the one or more computer processors, the read access rule for the stored data element to which access is requested; and

selectively permitting read access to the stored data element in response to the access request, based on the evaluation result for the read access rule.

3. The computer-implemented method of claim 2, wherein the operation further comprises:

defining, based on user input and for each of the stored data elements, a write access rule for the respective stored data element, wherein the write access rule comprises a condition for granting write access to the respective stored data element, the condition specifying a security level of the plurality of security levels and a category from the plurality of categories; wherein the condition for granting write access to one of the stored data elements differs from the condition for granting read access to the one of the stored data elements;

evaluating, by operation of the one or more computer processors, the write access rule for the stored data element to which access is requested; and

selectively permitting write access to the stored data element in response to the access request, based on the evaluation result for the write access rule.

4. The computer-implemented method of claim 3, wherein the condition for granting write access further specifies an exception for a user, whereby the exception allows write access to be granted to the user even if the condition is not satisfied.

5. The computer-implemented method of claim 3, wherein the condition of the write access rule is satisfied only if (i) the security level of the user meets the security level specified by the write access rule for the stored data element, and (ii) the category of the user matches the category specified by the write access rule for the stored data element.

6. The computer-implemented method of claim 1, wherein the condition for granting read access further specifies an exception for a user, whereby the exception allows read access to be granted to the user even if the condition is not satisfied.

7. The computer-implemented method of claim 1, wherein the condition of the read access rule is satisfied only if (i) the security level of the user meets the security level specified by the read access rule for the stored data element, and (ii) the category of the user matches the category specified by the read access rule for the stored data element.

8. A computer program product for controlling user access to stored data elements, the computer program product comprising a computer usable medium having computer usable program code configured to:

define, based on user input, an ordered plurality of security levels that describe sensitivity of the stored data elements;

define, based on user input, a plurality of categories that categorize the stored data elements;

associate each user with a security level from the ordered plurality of security levels and a category from the plurality of categories, thereby defining a security clearance for each respective user;

define, based on user input and for each of the stored data elements, a read access rule for the respective stored data element, wherein the read access rule comprises a condition for granting read access to the respective stored data element, the condition specifying a security level of the plurality of security levels and a category from the plurality of categories;

receive a user request to access a stored data element;

evaluate the read access rule for the stored data element to which access is requested; and

selectively permit read access to the stored data element in response to the access request, based on the evaluation result for the read access rule.

9. The computer program product of claim 8, wherein the computer usable program code is further configured to:

define, based on user input and for each of the stored data elements, a write access rule for the respective stored data element, wherein the write access rule comprises a condition for granting write access to the respective stored data element, the condition specifying a security level of the plurality of security levels and a category from the plurality of categories; wherein the condition for granting write access to one of the stored data elements differs from the condition for granting read access to the one of the stored data elements;

evaluate the write access rule for the stored data element to which access is requested; and

selectively permit write access to the stored data element in response to the access request, based on the evaluation result for the write access rule.

10. The computer program product of claim 9, wherein the condition for granting write access further specifies an exception for a user, whereby the exception allows write access to be granted to the user even if the condition is not satisfied.

11. The computer program product of claim 9, wherein the condition of the write access rule is satisfied only if (i) the security level of the user meets the security level specified by the write access rule for the stored data element, and (ii) the category of the user matches the category specified by the write access rule for the stored data element.

12. The computer program product of claim 8, wherein the condition for granting read access further specifies an exception for a user, whereby the exception allows read access to be granted to the user even if the condition is not satisfied.

13. The computer program product of claim 8, wherein the condition of the read access rule is satisfied only if (i) the security level of the user meets the security level specified by the read access rule for the stored data element, and (ii) the category of the user matches the category specified by the read access rule for the stored data element.

14. A system, comprising:

a processor; and

a memory containing an access control program, which when executed by the processor is configured to perform an operation for controlling user access to stored data elements, comprising:

defining, based on user input, an ordered plurality of security levels that describe sensitivity of the stored data elements;

defining, based on user input, a plurality of categories that categorize the stored data elements;

associating each user with a security level from the ordered plurality of security levels and a category from the plurality of categories, thereby defining a security clearance for each respective user;

defining, based on user input and for each of the stored data elements, a read access rule for the respective stored data element, wherein the read access rule comprises a condition for granting read access to the respective stored data element, the condition specifying a security level of the plurality of security levels and a category from the plurality of categories;

receiving a user request to access a stored data element;

evaluating the read access rule for the stored data element to which access is requested; and

selectively permitting read access to the stored data element in response to the access request, based on the evaluation result for the read access rule.

15. The system of claim 14, wherein the operation further comprises:

defining, based on user input and for each of the stored data elements, a write access rule for the respective stored data element, wherein the write access rule comprises a condition for granting write access to the respective stored data element, the condition specifying a security level of the plurality of security levels and a category from the plurality of categories; wherein the condition for granting write access to one of the stored data elements differs from the condition for granting read access to the one of the stored data elements;

evaluating the write access rule for the stored data element to which access is requested; and

selectively permitting write access to the stored data element in response to the access request, based on the evaluation result for the write access rule.

16. The system of claim 15, wherein the condition for granting write access further specifies an exception for a user, whereby the exception allows write access to be granted to the user even if the condition is not satisfied.

17. The system of claim 15, wherein the condition of the write access rule is satisfied only if (i) the security level of the user meets the security level specified by the write access rule for the stored data element, and (ii) the category of the user matches the category specified by the write access rule for the stored data element.

18. The system of claim 14, wherein the condition for granting read access further specifies an exception for a user, whereby the exception allows read access to be granted to the user even if the condition is not satisfied.

19. The system of claim 14, wherein the condition of the read access rule is satisfied only if (i) the security level of the user meets the security level specified by the read access rule for the stored data element, and (ii) the category of the user matches the category specified by the read access rule for the stored data element.

* * * * *

US007860875B2

(12) **United States Patent**
Bird et al.

(10) **Patent No.: US 7,860,875 B2**
(45) **Date of Patent: Dec. 28, 2010**

(54) **METHOD FOR MODIFYING A QUERY BY USE OF AN EXTERNAL SYSTEM FOR MANAGING ASSIGNMENT OF USER AND DATA CLASSIFICATIONS**

(75) Inventors: **Paul Miller Bird**, Markham (CA); **Walid Rjaibi**, Markham (CA)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 971 days.

(21) Appl. No.: **10/855,106**

(22) Filed: **May 26, 2004**

(65) **Prior Publication Data**

US 2005/0267865 A1 Dec. 1, 2005

(51) **Int. Cl.**
G06F 17/30 (2006.01)
G06F 21/24 (2006.01)

(52) **U.S. Cl.** **707/759**; 707/769; 707/783

(58) **Field of Classification Search** 707/1–10, 707/100–104.1

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,787,428	A	7/1998	Hart	707/9
6,487,552	B1 *	11/2002	Lei et al.	707/4
6,578,037	B1 *	6/2003	Wong et al.	707/10
6,581,060	B1	6/2003	Choy	707/9
2003/0154401	A1	8/2003	Hartman et al.	713/201
2003/0236782	A1 *	12/2003	Wong et al.	707/5
2004/0139043	A1 *	7/2004	Lei et al.	707/1
2005/0165799	A1 *	7/2005	Wong	707/100

OTHER PUBLICATIONS

Karjoth, G., *Access Control with IBM Tivoli Access Manager*, ACM Transactions on Information and System Security, vol. 6, No. 2, May 2003, pp. 232-257.

DeFazio, S., et al., *Integrating IR and RDBMS Using Cooperative Indexing*, SIGIR '95 Seattle WA, 1995 ACM 0-89791-714-6/95, pp. 84-92.

Winslett, et al., *Formal Query Languages for Secure Relational Databases*, ACM Transactions on Database Systems, vol. 19, No. 4., Dec. 1994, pp. 626-662.

Sandhu, R., *Access Control: Principles and Practice*, IEEE Communications Magazine, Sep. 1994, pp. 40-48.

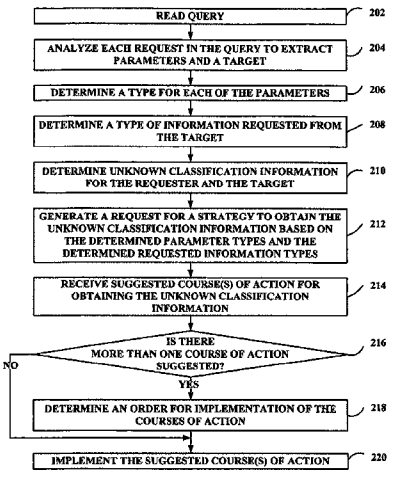
* cited by examiner

Primary Examiner—Tim T. Vo
Assistant Examiner—Sangwoo Ahn
(74) *Attorney, Agent, or Firm*—Sughrue Mion, PLLC

(57) **ABSTRACT**

Disclosed is a data processing-implemented method, a data processing system, and an article of manufacture for modifying a query during compilation of the query. The query includes a request for an element of data from a table in a database and parameters identifying the requested element. The data processing-implemented method includes determining available information from parameters for locating a classification of the requested element and a classification associated with the query, the requested data classification controlling access to the requested element according to the query associated classification, requesting a suggested action from an external system for obtaining a comparison of the requested data classification and the query associated classification based on the available information, receiving the suggested action from the external system responsive to the sent request, and incorporating the suggested action into the query, the suggested action effecting comparison of the requested data classification with the query associated classification.

14 Claims, 3 Drawing Sheets



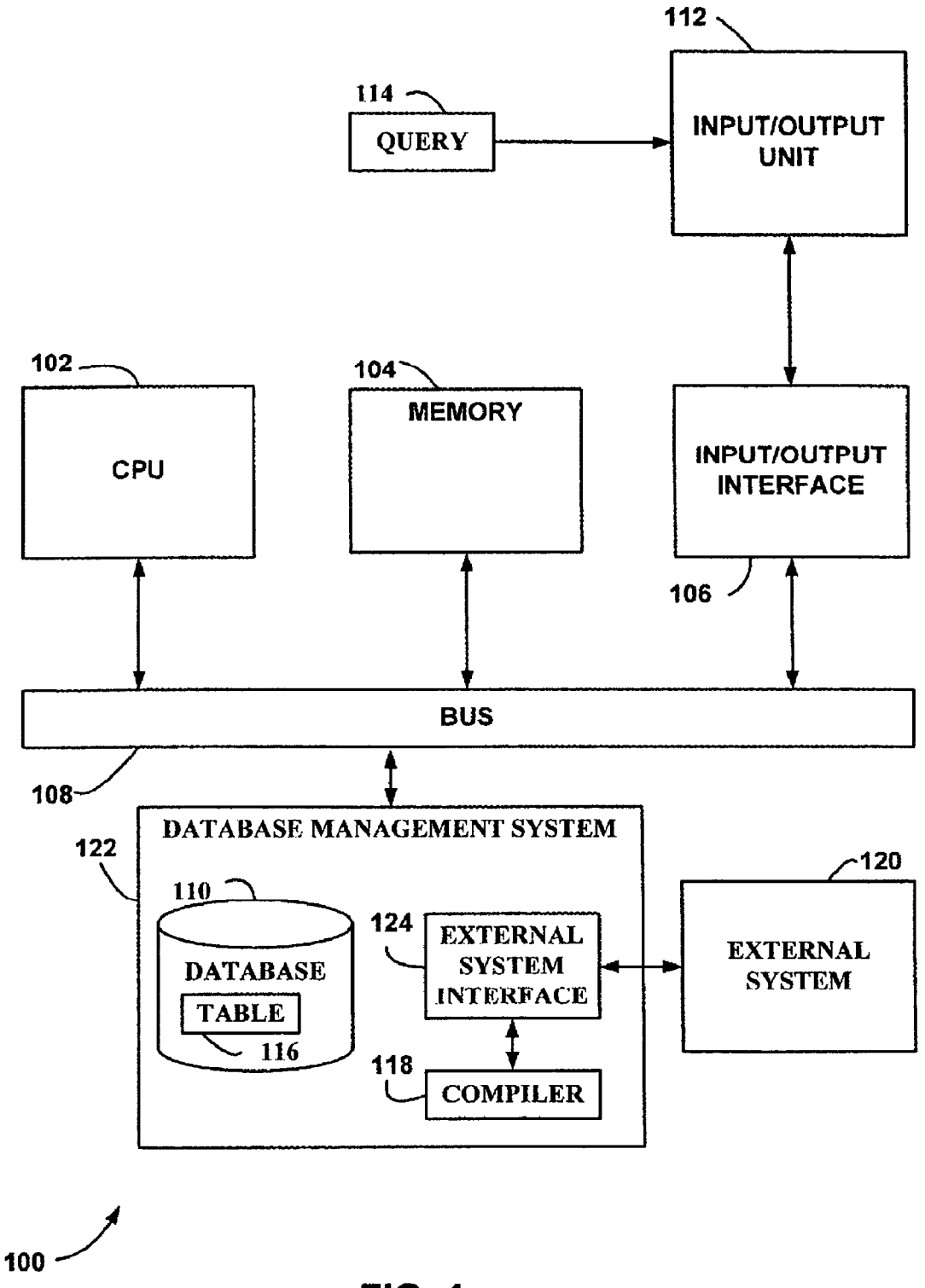

112
114
QUERY
INPUT/OUTPUT UNIT
102
CPU
104
MEMORY
INPUT/OUTPUT INTERFACE
106
BUS
108
DATABASE MANAGEMENT SYSTEM
122
110
DATABASE
TABLE
116
124
EXTERNAL SYSTEM INTERFACE
118
COMPILER
120
EXTERNAL SYSTEM
100

FIG. 1

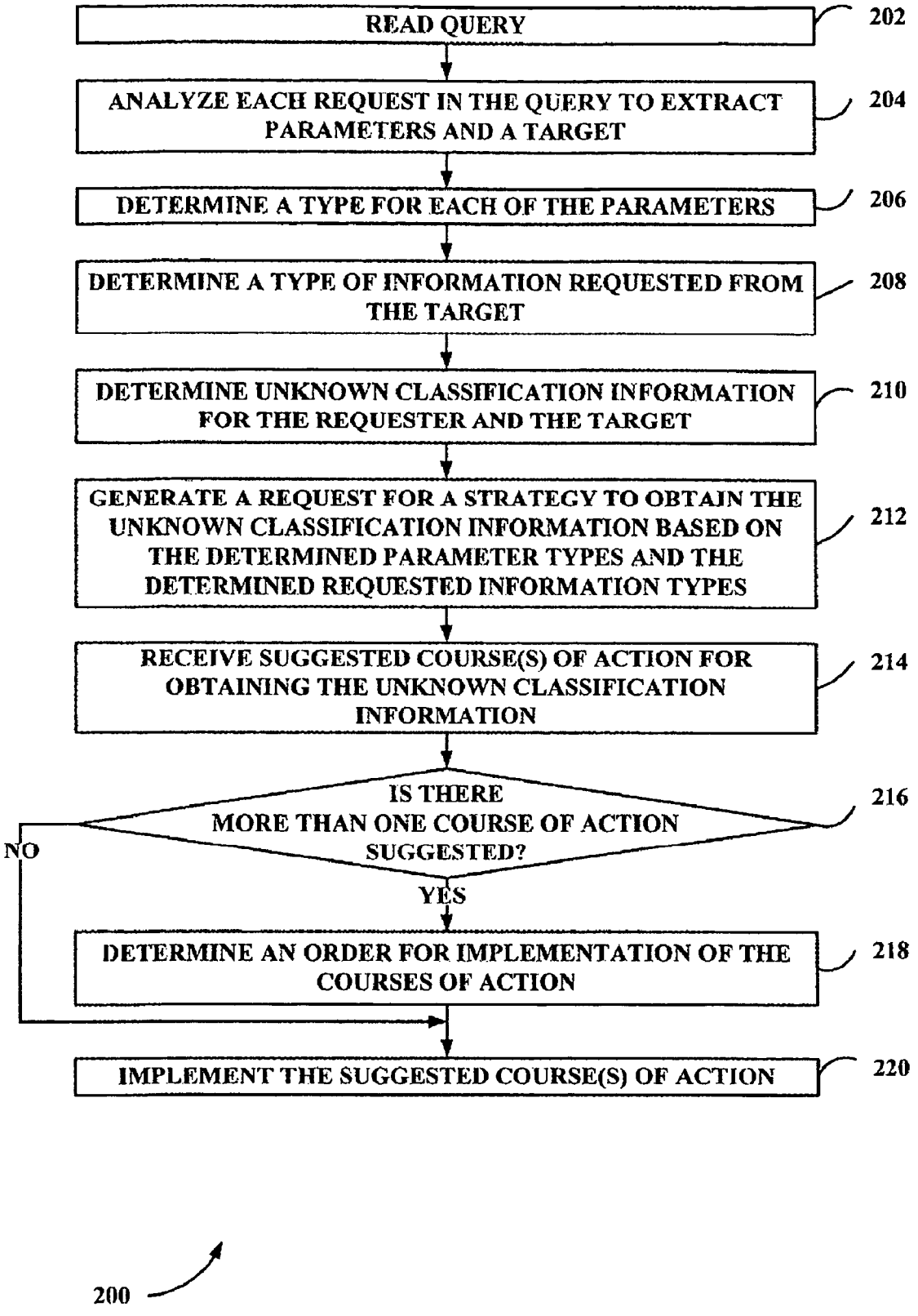

READ QUERY
202
ANALYZE EACH REQUEST IN THE QUERY TO EXTRACT PARAMETERS AND A TARGET
204
DETERMINE A TYPE FOR EACH OF THE PARAMETERS
206
DETERMINE A TYPE OF INFORMATION REQUESTED FROM THE TARGET
208
DETERMINE UNKNOWN CLASSIFICATION INFORMATION FOR THE REQUESTER AND THE TARGET
210
GENERATE A REQUEST FOR A STRATEGY TO OBTAIN THE UNKNOWN CLASSIFICATION INFORMATION BASED ON THE DETERMINED PARAMETER TYPES AND THE DETERMINED REQUESTED INFORMATION TYPES
212
RECEIVE SUGGESTED COURSE(S) OF ACTION FOR OBTAINING THE UNKNOWN CLASSIFICATION INFORMATION
214
IS THERE MORE THAN ONE COURSE OF ACTION SUGGESTED?
216
NO
YES
DETERMINE AN ORDER FOR IMPLEMENTATION OF THE COURSES OF ACTION
218
IMPLEMENT THE SUGGESTED COURSE(S) OF ACTION
220
200

FIG. 2

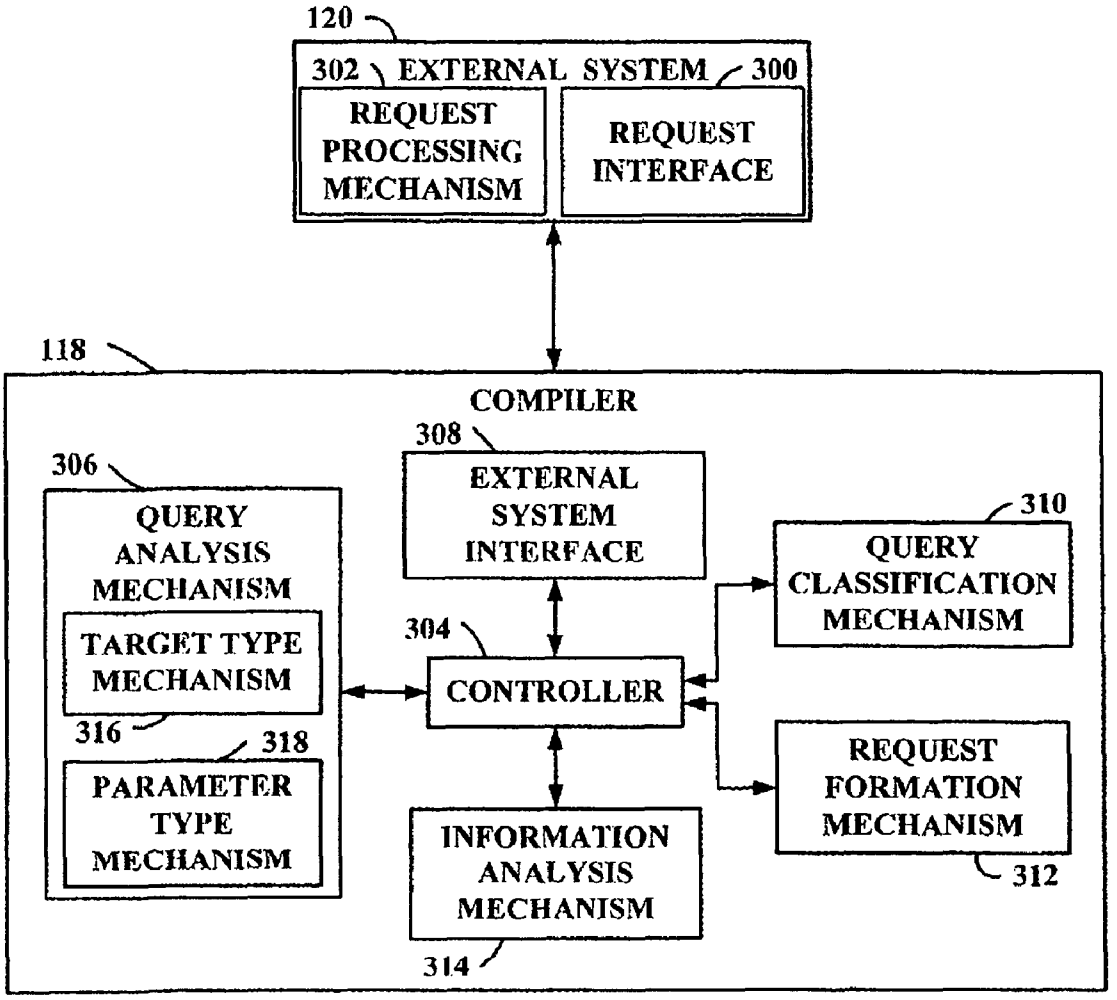

120
302
EXTERNAL SYSTEM
300
REQUEST PROCESSING MECHANISM
REQUEST INTERFACE
118
COMPILER
308
EXTERNAL SYSTEM INTERFACE
306
QUERY ANALYSIS MECHANISM
TARGET TYPE MECHANISM
316
318
PARAMETER TYPE MECHANISM
304
CONTROLLER
310
QUERY CLASSIFICATION MECHANISM
REQUEST FORMATION MECHANISM
312
INFORMATION ANALYSIS MECHANISM
314

FIG. 3

METHOD FOR MODIFYING A QUERY BY USE OF AN EXTERNAL SYSTEM FOR MANAGING ASSIGNMENT OF USER AND DATA CLASSIFICATIONS

FIELD OF THE INVENTION

The present invention relates to the field of database management systems, and more specifically, to a system, method and a computer program product for modifying a query by use of an external system for managing assignment of user and data classifications.

BACKGROUND

Information can be obtained from tables in a database using queries expressed in a database query language, such as Structured Query Language (SQL). The query is translated into an internal representation by a compiler of a database management system. This internal representation is interpreted by a runtime processor of the database management system to execute the query. Access to information in the database may be controlled according to a classification of both the tables and the user attempting to access the tables. For example, a user can only gain access to a specific table if the user's classification is such that access to the specific table is permitted based on the table's classification. The table's classification may be based on the entire table or on individual elements in each table (e.g. rows) with elements being classified to provide access to elements and not the entire table. The additional classifications produce complexities in classification management and tracking which may be governed by a system external to the database management system.

Compilers use various optimization techniques to minimize the time and computer resources used for execution of the internal representation of the query. The compiler determines an efficient access plan to satisfy the query by examining table information and related statistics. Controlling access to elements based on user and table classifications may involve integrating with an external system. Such integration during execution of the query often increases execution time, especially if such information is not readily available.

SUMMARY

In accordance with one aspect there is provided a data processing-implemented method for directing a data processing system to modify a query during compilation of the query, the query including a request for an element of data from a table in a database and parameters identifying the requested element, the data processing-implemented method including determining available information from parameters for locating a classification of the requested element and a classification associated with the query, the requested data classification controlling access to the requested element according to the query associated classification, requesting a suggested action from an external system for obtaining a comparison of the requested data classification and the query associated classification based on the available information, receiving the suggested action from the external system responsive to the sent request, and incorporating the suggested action into the query, the suggested action effecting comparison of the requested data classification with the query associated classification.

In accordance with another aspect there is provided a data processing system for modifying a query during compilation of the query, the query including a request for an element of data from a table in a database and parameters identifying the requested element, the data processing system including a query analysis mechanism for determining available information from parameters for locating a classification of the requested element and a classification associated with the query, the requested data classification controlling access to the requested element according to the query associated classification, a request mechanism for preparing a request to the external system, the request asking the external system to provide a suggested action for obtaining a comparison of the requested data classification and the query associated classification, the request comprising the available information, an external system interface for requesting a suggested action from an external system for obtaining a comparison of the requested data classification and the query associated classification based on the available information, and receiving the suggested action from the external system responsive to the sent request, and a modification mechanism for incorporating the suggested action into the query to effect comparison of the requested data classification with the query associated classification.

In accordance with a further aspect there is provided an article of manufacture for directing a data processing system to modify a query during compilation of the query, the query including a request for an element of data from a table in a database and parameters identifying the requested element, the article of manufacture including a program usable medium embodying one or more executable data processing system instructions, the executable data processing system instructions including executable data processing system instructions for determining available information from parameters for locating a classification of the requested element and a classification associated with the query, the requested data classification controlling access to the requested element according to the query associated classification, executable data processing system instructions for requesting a suggested action from an external system for obtaining a comparison of the requested data classifications and the query associated classification based on the available information, executable data processing system instructions for receiving the suggested action from the external system responsive to the sent request, and executable data processing system instructions for incorporating the suggested action into the query, the suggested action effecting comparison of the requested data classification with the query associated classification.

Other aspects and features of the present invention will become apparent to those ordinarily skilled in the art upon review of the following description of specific embodiments of the invention in conjunction with the accompanying figures.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be described in conjunction with the drawings in which:

FIG. **1** is an exemplary computing environment in which a database management system (DBMS) may be actualized;

FIG. **2** illustrates operations of a compiler of the DBMS of FIG. **1** for modifying a query based on information from an external system; and

FIG. **3** illustrates functional components of the compiler in the DBMS of FIG. **1** for modifying a query based on information from the external system.

DETAILED DESCRIPTION OF EMBODIMENTS

The following detailed description of the embodiments do not limit the implementation of the embodiments to any particular computer programming language. The computer program product may be implemented in any computer programming language provided that the operating system provides the facilities that support the requirements of the computer program product. A preferred embodiment is implemented in the C or C++ computer programming language (or may be implemented in other computer programming languages in conjunction with C/C++). Any limitations presented would be a result of a particular type of operating system, computer programming language, or data processing system and would not be a limitation of the embodiments described herein.

FIG. **1** illustrates a configuration of a computing environment **100** comprising a data processing system **126** in which an embodiment of a database management system **122** may be implemented.

The data processing system **126** includes a central processing unit (CPU) **102**, a memory **104**, an input/output interface **106** and a bus **108**. The CPU **102**, the memory **104** and the input/output interface **106** are connected with one another via the bus **108**. The input/output interface **106** is configured so that it can be connected to an input/output unit **112** in the computing environment **100**.

The CPU **102** can be a commercially available CPU or a customized CPU suitable for operations described herein. Other variations of the CPU **102** can include a plurality of CPUs interconnected to coordinate various operations and functions. The data processing system **126** serves as an apparatus for performing the present method by the CPU **102** executing the present invention.

Data and instructions that are to be executed by the CPU **102** reside in the memory **104**. The memory **104** contains a database management system (DBMS) **122** and a database **110** with multiple tables **116** (only one table is shown for illustration purposes) that hold information. The instructions are internal representations of programs that run on the data processing system **126**, such as the database management system **122**. The programs operate on the data. For example, if the program is the database management system **122**, the data can be rows in the table **116**. The database management system **122** comprises a compiler **118** and an external system interface **124**. The database management system **122** retains an indication of operating conditions, such as an identifier for the user who submitted the query **114**, when compiling and executing the query **114**.

The information in the tables **116** may be accessed by a query **114** that is received by the input/output unit **112** and is retained in the memory **104**. The query **114** may be presented in an SQL format that is compiled by the compiler **118** to form an internal representation that is interpreted for execution. The present invention may be embodied in the compiler **118**. Alternatively, the present invention may be provided as an extension of the functionality of the compiler **118**. The present invention may be embodied in a program stored in, for example, the memory **104**. Alternatively, the present invention may be recorded on any type of recording medium such as a magnetic disk or an optical disk. The present invention recorded on such a recording medium is loaded to the memory **104** of the data processing system **126** via the input/output unit **112** (e.g. a disk drive).

The compiler **118** recognizes statements in the query **114** including keywords that represent commands and relevant arguments. During the formation of the internal representation from the query **114**, the compiler **118** modifies the query **114** to improve performance during execution.

The external system interface **124** is in communication with an external system **120** while modifying the query **114** to improve performance. The external system **120** may be a system external to the database management system **122** but residing in the data processing system **126** or it may be external to the data processing system **126**. In the latercase the external system **120** may communicate with the data processing system **126** via a direct link or through a communications network.

The external system **120** manages the assignment of classifications to users and sections of data in the tables **116**. For example, given a user identification and a table name, the external system **120** knows how to obtain the classification of that user. The external system **120** contains a classification that is associated with the query **114** which may be a classification for a user identification from which the query **114** was submitted, a classification of the location from which the query **114** was submitted or some other such similar classification basis. The external system **120** may also contain access rules that govern when a user with a particular classification can access an element from the table **116**. An interface in the external system **120** accepts questions from the compiler **118** providing available information and desired information. Through this interface the external system **120** is able to either provide the information requested by the compiler **118** or provide a course of action for obtaining the information.

As a result, the query **114** may be modified to include information on a user's classification or table classification or a comparison of the two classifications. Such information might be determined by interfacing with the external system **120** during execution or such interfacing may be performed in advance by the compiler **118** and the results incorporated into the query **114**.

FIG. **2** illustrates operations of the compiler **118** that modify the query **114** based on information from the external system **120**. The query **114**, containing at least one request for information, is read in step **202**. Each request is analyzed in step **204** to extract parameters of the request and a target of the request. The type of information of each of the parameters is determined in step **206**. The parameters may also include an identification associated with the query **114** such as a user who submitted the query **114** or a location of the submission. The type of information requested from the target is determined in step **208**.

The table **116** in the database **110** contains information that is classified. In order for a user who submitted the query **114** to obtain the requested information from the table **116**, access rules for the table **116** in the external system **120** might indicate that the user have a classification that corresponds to the classification of the requested information. Based on the types of parameters included in the request, the user's identification (as contained in the database management system **122**) and the type of information requested, step **210** determines what information regarding the user's classification and the information's classification is unknown.

Since this unknown classification information is determined prior to completing execution of the query **114**, the compiler **118** determines how the classification information can be determined in conjunction with the external system **120**. A request for a strategy to obtain the unknown classification information is generated in step **212**. This request is

based on the determined types of the parameters and the determined type of the requested information.

Suggested course(s) of action for obtaining the unknown classification information are received in step **214** from the external system **120**. If there are multiple types of unknown classification information then there may be multiple courses of action that will be received. Each course of action received may be directed to obtaining one of the types of unknown classification information.

If there is more than one course of action received, as determined in step **216**, then an order for the courses of action is determined in step **218**. This order may depend on information required by each course of action. For example, if one course of action uses information dependent on a second course of action then the second course of action is implemented first. The order for implementation of the courses of action may optionally be supplied with the suggested courses of action

The course(s) of action are inserted into the query in step **220** such that they can be easily implemented during execution.

FIG. **3** illustrates functional components of the compiler **118** in the DBMS **122** for modifying the query **114** based on information from the external system **120**. The external system **120** includes a request interface **300** and a request processing mechanism **302**. The compiler **118** includes a controller **304** in communication with a query analysis mechanism **306**, a management interface **308**, an information analysis mechanism **314**, a query classification mechanism **310** and a request formation mechanism **312**.

The controller **304** in the compiler **118** manages compiling the query **114** in order to form an internal representation thereof. During compiling, the controller **304** coordinates modification of the query **114** to improve execution performance. When the controller **304** detects a request in the query **114** that requires a check of a user's classification with the classification of requested information, the request is provided to the query analysis mechanism **306**.

The query analysis mechanism **306** includes a target type mechanism **316** and a parameter type mechanism **318** that collectively function to determine the information defining the request and the information sought from the request. The parameter type mechanism **318** extracts the type of information of the parameters that define the request. The target type mechanism **316** determines the type of information that has been requested. The query analysis mechanism **306** provides the parameters and target types to the controller **304** where it is passed to the information analysis mechanism **314** and the request formation mechanism **312**.

The information analysis mechanism **314** receives the parameters and target types and assesses what information is available for determining the requested information classification and user's classification. Based on the available information, the information analysis mechanism **314** determines the information that is unknown that is to be used for completing data access qualification for the user. The determined unknown information is provided to the controller **314** from which it is passed to the request formation mechanism **312**.

The request formation mechanism **312** receives the parameters and target types as well as an indication of the unknown information to be used in determining the requested information classification and user's classification. The request formation mechanism **312** formulates a request on how to obtain the unknown information based on the parameters and target types. This request is provided to the controller **304** to be passed to the management interface **308**. The external system interface **308** provides the request to the external system interface **124** in the database management system **122** so that the request can be submitted to the external system **120**.

The request from the request formation mechanism **312** may be one or a combination of, for example:

Q1: Given an user identification and a table name, how can the user classification be obtained?

Q2: Given a set of data values and a table name, how can the element classification be obtained?

Q3: Given a user classification and an element classification, how can the two be compared?

The request interface **300** of the external system **120** receives the request from the compiler **118**. The request is provided to the request processing mechanism **302** where a knowledge base may be drawn upon to produce suggestions regarding the manner in which the unknown information can be obtained. If there are multiple courses of action then the suggestion may involve multiple courses of action, each pertaining to obtaining a different piece of unknown information. The suggestions from the request processing mechanism **302** are provided to the controller **304** via the request interface **300** and the management interface **308** through the external system interface **124**.

Based on the above exemplary requests, the suggested course(s) of action form the request processing mechanism **302** may be one or a combination of, for example:

A1: A subquery which can be used to select a user classification or element classification from a table in the database **110** known to the external system **120**.

A2: A predicate which can be used to filter out the table's elements (rows or columns) that have a classification that do not match the user's classification.

A3: A set of values presented directly or indirectly via a session variable or special register. These values can represent a set of user classification or a set or element classifications.

A4: A query which can be used to generate an internal mapping table for use by the executable form of the query **114**. For a given table, the mapping table enables identification of the classification of an element in the table. For example, the mapping table may consist of (n+1) columns where the first n columns represent the table columns from which to derive the element classifications and the last columns represents a classification level. When such a mapping table was not previously created, an internal mapping table for use by the executable form of the query **114** can be generated for this purpose.

A5: A request to call the external system at execution time of the query **114** for classification information.

The above requests may produce the suggested course(s) of action as indicated below:

Action on Q1:

A1: A subquery that can be used to select the user classification for the user identification from a database table known to the external system **120**.

A3: A data value(s) that indicates the user classification for the user identification.

A5: An indication that the external system **120** should be asked for this information at execution time.

Action on Q2:

A1: A subquery that can be used to select the element classification for the current element from a mapping table known to the external system **120**.

A4: A query that can be used to generate an internal mapping table for use by the executable form of the query **114**.

A5: An indication that the external system **120** should be asked for this information at execution time.

Action on Q3:

A2: A predicate that the compiler **118** can add to the query **114** to filter out the table's elements that do not match the user's classification. The general form of predicates returned will be an IN predicate but inequality predicates are also possible, particularly if the element classification or user's classification represent a hierarchy. This type of advice is most likely to be returned when the element classification is stored within the table itself or when a mapping table has been created. If the element classification is stored within the element then the predicate will refer to the table's column where the element classification level is stored, otherwise, the predicate will refer to the mapping table's column where the classification level is stored.

A3: A set of values representing the element classification allowed for the given user's classification. Element level access control may then be enforced by, for example:

1. The compiler **118** altering the query **114** to add a predicate using the set of values received. This choice is possible if the element classification is stored within the table itself or a mapping table has been created.
2. If the element classification is not stored within the table and a mapping table has not been created then a predicate cannot be used. In this case, interaction between the DBMS **122** and the external system **120** is used during execution of the query **114** to enforce element access control. For each element accessed, the data in the set of columns defining the element classification and the full table is submitted to the external system **120** with the result being the element classification. The result is compared against the set of values for the given user classification to determine if the element can be viewed or altered by that user. To reduce the number of times the DBMS **122** makes a call to the external system **120** to obtain the element classification a caching technique may be used. For example, the information that could be stored in the cache may be the full table name, the data defining the element classification and the element classification as returned by the external system **120**.

A5: An indication that the external system **120** should be asked for this information at execution time.

The request provided by the request processing mechanism **302** may also provide an indication of whether or not the suggested course(s) of action can be used of all users or only for a provided user identification.

The controller **304** provides the suggested course(s) of action to the query classification mechanism **310** where an order is determined for the course(s) of action based on dependence of the results of each course of action. Alternatively, this order may be specified by the external system **120** and received with the suggested course(s) of action. After the order has been determined, the query classification mechanism **310** modifies the query **114** to include the course(s) of action.

The following are examples of modifying an SQL query to include obtaining classification information.

A table T1 (C1, C2, C3, . . . , Cn) represents a table where the classification level of an element and the user classification is an element of the ordered set S={TOP SECRET, SECRET, CONFIDENTIAL, CLASSIFIED, UNCLASSIFIED}. The element level access control policy for this example states that an element with a classification r can be viewed by a user with a classification u only if u>=r. Suppose that a user with a classification level 'CONFIDENTIAL' issues a query SELECT * FROM T1.

Scenario 1

The compiler **118** sends a request corresponding with Q1 from above to the external system **120** to obtain the user's classification. Suppose the external system **120** provides a suggested course of action corresponding with A3 from above; that is, a data value representing the user's classification. The compiler **118** then submits a second request to the external system **120** based on Q3 above by submitting the table name (T1) and the user's classification (CONFIDENTIAL). Suppose the external system **120** returns a suggested course of action corresponding with A2 from above. That is, in response to the second request the external system **120** returned a predicate in, for example, "C1 IN ('CONFIDENTIAL', 'CLASSIFIED', 'UNCLASSIFIED')". Based on the received courses of action the compiler **118** modifies the query **114** to incorporate the predicate providing a query such as

```
SELECT * FROM T1 WHERE C1 IN ('CONFIDEN-
    TIAL', 'CLASSIFIED', 'UNCLASSIFIED').
```

Given that the set is ordered and represents a hierarchy, the predicate returned could also be "C1>'CONFIDENTIAL'".

Scenario 2

Suppose an element classification is determined based on the values in columns C1 and C2 as follows:

C1	C2	Element Classification
1	1	TOP SECRET
2	2	SECRET
3	3	CONFIDENTIAL
4	4	CLASSIFIED
5	5	UNCLASSIFIED

The compiler **118** sends the external system **120** a request corresponding with request Q1 to obtain the user's classification. Suppose the suggest course of action is A3; that is, a data value represent the user's classification. The compiler **118** submits a second request based on the table name (T1) and the set of column names defined in the classification mapping shown above (Cl and C2). The suggested course of action in response to the second request depends on whether a mapping table exists.

Response 1: A Mapping Table Exists

A database table (T1MAP) storing mapping information has been created and is known to the external system **120**. T1MAP consists of three columns, namely, C1, C2 and LEVEL. For each pair of values (C1, C2), the LEVEL column indicates an element classification. Based on this information, the external system **120** can return A1 as the suggested course of action; that is, a subquery to select an element classification from T1MAP. The subquery would be as follows:

```
SELECT LEVEL FROM T1MAP WHERE
    T1MAP.C1=T1.C1 AND T1MAP.2=T1.C1.
```

The compiler **118** then sends a request to the external system **120** corresponding with request Q3 by submitting the table name (T1) and the user's classification (CONFIDENTIAL). If the external system **120** returns suggested action A2, then the predicate returned would be: "T1MAP.LEVEL>'CONFIDENTIAL'". Based on the sug-

gested course of action the compiler **118** modifies the query **114** to incorporate the predicate and subquery. The modified query would be:

```
SELECT * FROM T1, T1MAP WHERE
    (T1.C1=T1MAP.C1 AND T1.C2=T1MAP.C2)
    AND (T1MAP.LEVEL> 'CONFIDENTIAL').
```

Response 2: A Mapping Table Does Not Exist

If a mapping table does not exist then the suggested course of action provided to the compiler **118** might be action A5, an indication to submit the same request during execution. The second request submitted by the compiler **118** corresponds with request Q3 and submits the table name (T1) and the user's classification (CONFIDENTIAL). The external system may provide action A3, a set of data values representing the element classifications allowed for the user (i.e. all elements having 'CONFIDENTIAL', 'CLASSIFIED', and 'UNCLASSIFIED'). Based on the suggestion course of action the compiler **118** does not modify the query **114** but inserts logic into the internal representation to perform the following tasks:

- For each element obtained, call the external system **120** by submitting the table name (T1) and the values (C1,C2).
- Obtain the element classification from the call to the external system **120**.
- If the element classification is an element of the set {'CONFIDENTIAL', 'CLASSIFIED', 'UNCLASSIFIED'} then include the element in the result set; otherwise, discard the element.

Although the classification of the user is used as the basis for obtaining the requested element of data, any classification associated with the query **114** may be used. Such other associated classifications may include a classification of the location from which the query **114** was submitted.

The elements of data that are accessed may be the rows of the tables **116** or the columns of the tables **116** or some other delineation of portions of the tables **116**.

It will be appreciated that the elements described above may be adapted for specific conditions or functions. The concepts of the present invention can be further extended to a variety of other applications that are clearly within the scope of this invention. Having thus described the present invention with respect to preferred embodiments as implemented, it will be apparent to those skilled in the art that many modifications and enhancements are possible to the present invention without departing from the basic concepts as described in the preferred embodiment of the present invention. Therefore, what is intended to be protected by way of letters patent should be limited only by the scope of the following claims.

The invention claimed is:

1. A data processing-implemented method for directing a data processing system to modify a query during compilation of the query, the query comprising a request for an element of data from a table in a database and parameters identifying the requested element, the data processing-implemented method comprising:

- determining, by a computer, available information from parameters for locating a classification of the requested element and a classification associated with the query, the requested data classification controlling access to the requested element according to the query associated classification;
- requesting a suggested action from an external system for obtaining a comparison of the requested data classification and the query associated classification based on the available information;
- receiving the suggested action from the external system responsive to the sent request; and
- incorporating the suggested action into the query, the suggested action effecting comparison of the requested data classification with the query associated classification,
- wherein if the external system knows the requested data classification and the query associated classification, the suggested action is provided prior to execution of the query, and comprises at least one of the requested data classification, the query associated classification, a course of action for obtaining the requested data classification, a course of action for obtaining the query associated classification, and a comparison of the requested data classification and the query associated classification, and
- wherein if the external system does not know the requested data classification and the query associated classification, the suggested action comprises a request to call the external system at execution time of the query.

2. The data processing-implemented method according to claim **1** wherein the requesting the suggested action comprises:

- determining unknown information used to obtain a comparison of the requested data classification with the query associated classification; and
- sending a request to the external system for the suggested action, the suggested action pertaining to obtaining the unknown information.

3. The data processing-implemented method according to claim **1** wherein the requesting the suggested action comprises:

- selecting a request from one of a plurality of formulated requests based on the available information; and
- sending the selected request to the external system to obtain the suggested action.

4. The method according to claim **3** wherein the selecting comprises:

- selecting the request from the plurality of formulated requests consisting of:
  - a request for the query associated classification based on providing an identifier for the table and an identifier associated with the query, and
  - a request for the requested data classification based on providing an identifier for the requested element and the table identifier, and a request for a comparison of the requested data classification with the query associated classification.

5. The data processing-implemented method according to claim **2** wherein the incorporating the suggested action comprises:

- incorporating a subquery into the query to obtain unknown information from a table.

6. The method according to claim **1** wherein the incorporating the suggested action comprises:

- incorporating a predicate into the query to delimit sections of the table that can be obtained by the query according to the requested data classification and the query associated classification.

7. The data processing-implemented method according to claim **1** wherein the incorporating the suggested action comprises any one of:

- incorporating a set of values into the query representing the unknown information;
- incorporating a second query into the query to generate a mapping table mapping classifications to elements of data in the table; and

incorporating a request to the external system to be sent during execution of the query.

8. The data processing-implemented method according to claim **1** wherein the comparison comprises:

comparing the data classification with the query associated classification comprising the suggested action and determining if the query associated classification is equal to or greater than the data classification.

9. The data processing-implemented method for directing a data processing system according to claim **1**, wherein the external system is external to and functions independently from the data processing system, and communicates with the data processing system through a communications network.

10. The data processing-implemented method for directing a data processing system according to claim **1**,

wherein the external system contains classification information and access rules that govern access to data according to a particular classification, and

wherein the external system is separate of the data processing system.

11. The data processing-implemented method according to claim **1**, wherein the query associated classification is based on a classification of a user submitting the query, and the requested data classification is based on a permission level of a user authorized to view the requested data.

12. The data processing-implemented method according to claim **11**, wherein the suggested action includes at least one of instructions for obtaining the requested data classification when the requested data classification cannot be obtained with information in the query, and instructions for obtaining the query associated classification when the query associated classification cannot be obtained with information in the query.

13. The data processing-implemented method according to claim **12**, wherein the requested data classification is obtained using the instructions for obtaining the requested data classification, and the query associated classification is obtained using the instructions for obtaining the query associated classification and are provided for the comparison of the obtained requested data classification with the obtained query associated classification.

14. The data processing-implemented method according to claim **1**, wherein when the external system is called at the execution time of the query, the query is modified during execution of the query.

* * * * *

US 20090063951A1

(19) **United States**

(12) **Patent Application Publication**

(10) **Pub. No.: US 2009/0063951 A1**

(43) **Pub. Date: Mar. 5, 2009**

Rjaibi et al.

(54) **FINE-GRAINED, LABEL-BASED, XML ACCESS CONTROL MODEL**

(75) Inventors: **Walid Rjaibi**, Markham (CA); **Zheng (Alex) Zhang**, Toronto (CA)

Correspondence Address:
Kunzler & McKenzie
8 EAST BROADWAY, SUITE 600
SALT LAKE CITY, UT 84111 (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(21) Appl. No.: **11/849,267**

(22) Filed: **Sep. 1, 2007**

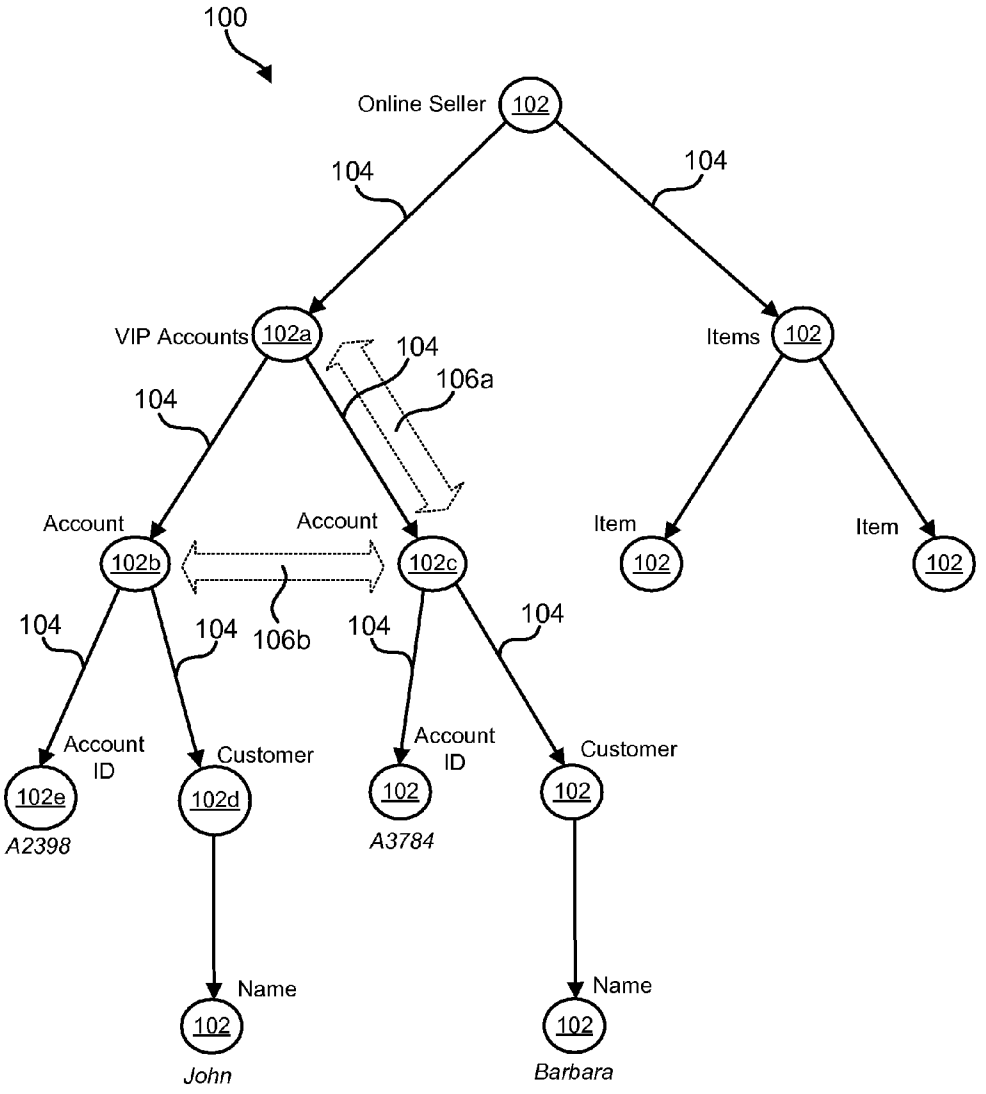
Publication Classification

(51) **Int. Cl.**
G06F 15/00 (2006.01)

(52) **U.S. Cl.** **715/234**

(57) **ABSTRACT**

A method for controlling access to an XML document includes referencing a schema definition comprising a path security label definition associated with a sibling-to-sibling path of an XML document. An XML document may then be validated by comparing it with the schema definition. This validation may include verifying that the XML document has a path security label associated with a sibling-to-sibling path that is at least as restrictive as that specified by the path security label definition. An access security label may be assigned to a user seeking to access the sibling-to-sibling path. The path security label and the access security label may then be compared, using pre-determined access rules, to determine whether the user is authorized to access the sibling-to-sibling path. Access to the sibling-to-sibling path may then be granted or denied according to the access rules.



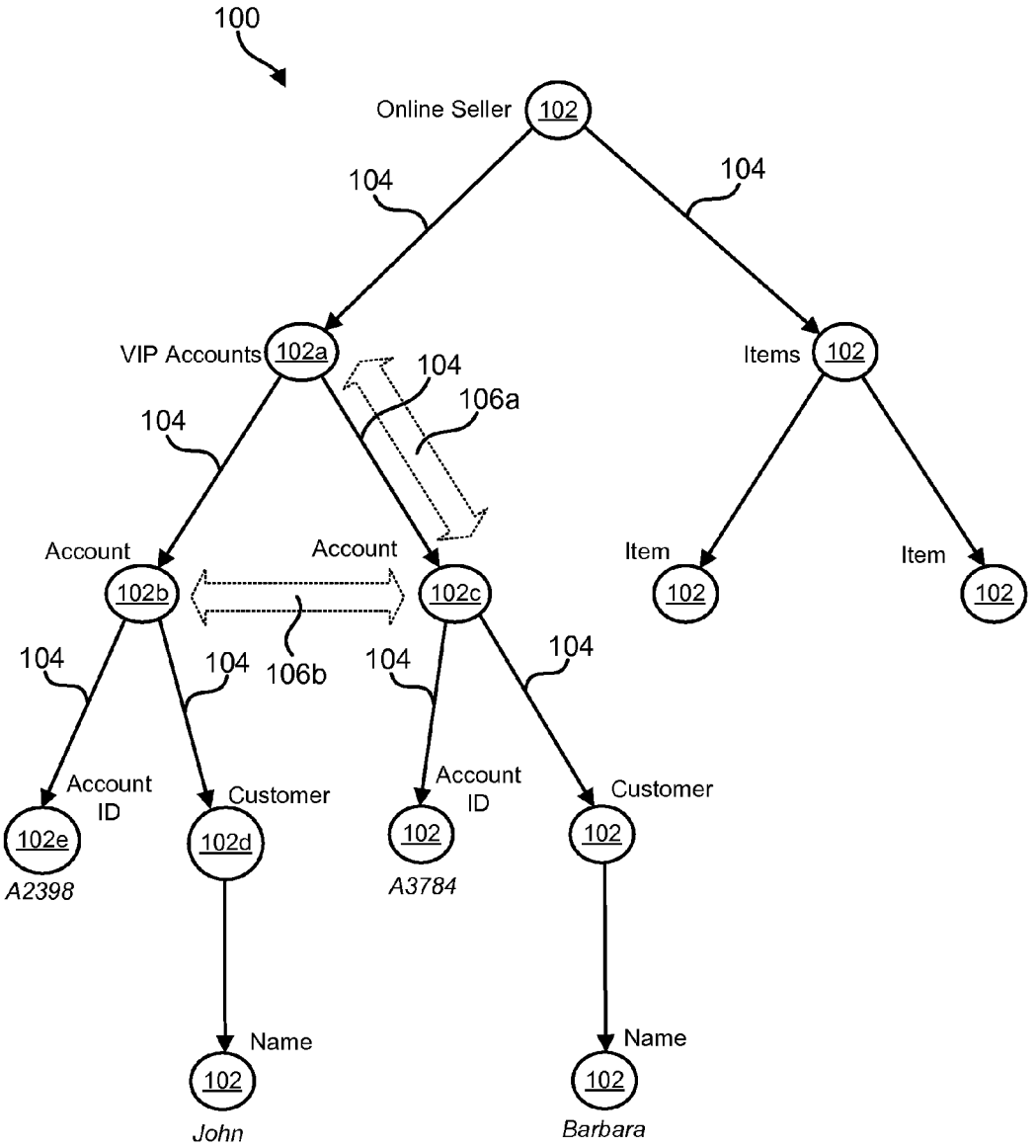


Fig. 1

ATTACH *path-label*
ANCS *path1*
DESC *path2*

Fig. 2

ATTACH *path-label*
NODE *path1*
PRECEDING-SIBLING *path2*
FOLLOWING-SIBLING *path3*

Fig. 3

FINE-GRAINED, LABEL-BASED, XML ACCESS CONTROL MODEL

BACKGROUND OF THE INVENTION

[0001] 1. Field of the Invention

[0002] This invention relates to XML access control and more particular to fine-grained, label-based, XML access control models.

[0003] 2. Description of the Related Art

[0004] XML has rapidly emerged as the prevalent standard for representing and exchanging business and other sensitive data over the Internet. The current trend to add XML support to database systems, however, poses new security challenges in an environment where both relational and XML data coexist. In particular, fine-grained access control methodologies may be even more important for XML data than for relational data, given the more flexible and less homogeneous structure of XML data compared to relational tables and rows.

[0005] Controlling access to XML data may be more difficult than controlling access to relational data for several reasons. First, the semi-structured nature of XML data, where a schema may be absent, or, even if present, may allow significantly more flexibility and variability in the structure of the document than is allowed by a relational schema. Second, the hierarchical structure of XML may require specifying how access privileges to certain nodes propagate to and from the nodes' ancestors and descendants.

[0006] In almost all models for controlling access to XML, the smallest unit of protection is a node of an XML document, which is typically specified using an XPath fragment. Access to ancestor/descendant and sibling relationships among nodes has typically not been considered. In general, an access control policy consists of positive or negative authorization rules that grant or deny access to selected nodes of an XML document. The main difference between most XML access control models lies in privilege propagation. For example, some models forbid access to entire sub-trees that are rooted at inaccessible nodes.

[0007] In other models, an ancestor node for which access is denied may be masked as an empty node if access is granted to a descendant node. However, this model may make the literal of the forbidden ancestor visible in the path from the root node to the authorized node. In some cases, this situation may be improved by replacing the literal of an ancestor node literal with a dummy value. However, this still does not solve the problem that different descendant nodes may require their ancestor's literal to be visible or invisible in a different manner. Accordingly, each of the above models makes it difficult to define a view that precisely describes the path leading to an authorized node.

[0008] In view of the foregoing, what is needed is an access control model for XML that provides a more fine-grained level of control. Ideally, such a model would be able to protect relationships between nodes as opposed to the nodes themselves. Further needed is a model that utilizes security labels to protect these relationships.

SUMMARY OF THE INVENTION

[0009] The present invention has been developed in response to the present state of the art, and in particular, in response to the problems and needs in the art that have not yet been fully solved by currently available methods for controlling access to information in XML documents. Accordingly, the present invention has been developed to provide a fine-grained, label-based model for controlling access to XML documents that remedies various problems in the art.

[0010] Consistent with the foregoing and in accordance with the invention as embodied and broadly described herein, a method for controlling access to an XML document includes referencing a schema definition comprising a path security label definition associated with a path of an XML document. As used herein the term "path" in an XML document refers to an ancestor-to-descendent path, a sibling-to-sibling path such paths, edges, and relationships between nodes of an XML document. An XML document with security labels may then be validated by comparing it with the schema definition. This validation may include verifying that the XML document has a path security label associated with a path that is at least as restrictive as that specified by the path security label definition. Similarly, an access security label may be defined for a user seeking to access a sibling-to-sibling path. In one embodiment, the security administrator may define the access security label for a user. The path security label and the access security label may be compared, using pre-determined access rules, to determine whether the user is authorized to access the sibling-to-sibling path. Access to the sibling-to-sibling path may then be granted or denied according to the access rules.

[0011] In a second aspect of the invention, a computer program product may be provided to control access to an XML document comprising a plurality of nodes and a plurality of paths, or relationships, between the nodes. The computer program product may include a computer-readable medium storing a program of computer-readable instructions. When executed, these instructions may cause a computer to generate a schema definition comprising a path security label definition associated with a sibling-to-sibling path of an XML document. The instructions may further enable an XML document to be validated by comparing it with the schema definition. This validation may include verifying that the XML document has a path security label associated with a sibling-to-sibling path that is at least as restrictive as that specified by the path security label definition. These instructions may further cause the computer to reference an access security label to a user seeking to access the sibling-to-sibling path of the XML document and compare, using pre-determined access rules, the path security label to the access security label to determine whether the user is authorized to access the sibling-to-sibling path. In one embodiment, these instructions may cause the computer to assign an access security label to an XML document that fails to comply with a given Document Type Definition (DTD) or XML Schema Definition (XSD). The access security label assigned may be at least as restrictive as a path security label designated in the DTD or XSD. Finally, the instructions may cause the computer to grant or deny access to the sibling-to-sibling path according to the access rules. The present invention provides novel methods for controlling access to XML documents. The features and advantages of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] In order that the advantages of the invention will be readily understood, a more particular description of the invention briefly described above will be rendered by reference to

specific embodiments illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered limiting of its scope, the invention will be described and explained with additional specificity and detail through use of the accompanying drawings, in which:

[0013] FIG. **1** illustrates one embodiment of an XML document tree structure that includes multiple nodes and paths between the nodes;

[0014] FIG. **2** illustrates one embodiment of an SQL/XPath extension, or statement, to attach a path security label to a parent-to-child path; and

[0015] FIG. **3** illustrates one embodiment of an SQL/XPath extension, or statement, to attach a path security label to a sibling-to-sibling path.

DETAILED DESCRIPTION OF THE INVENTION

[0016] It will be readily understood that the components of the present invention, as generally described and illustrated in the Figures herein, could be arranged and designed in a wide variety of different configurations. Thus, the following more detailed description of the embodiments of systems and methods in accordance with the present invention, as represented in the Figures, is not intended to limit the scope of the invention, as claimed, but is merely representative of certain examples of presently contemplated embodiments in accordance with the invention. The presently described embodiments will be best understood by reference to the drawings, wherein like parts are designated by like numerals throughout.

[0017] Referring to FIG. **1**, one embodiment of an XML document tree structure **100** is illustrated to provide a basic understanding of the invention. Here, the document tree structure **100** stores account and item information associated with an online seller. As shown, the document tree structure **100** includes a plurality of nodes **102** arranged in a hierarchical tree structure. The relationship between the nodes **102** may be represented by a plurality of paths **104** traveling between each of the nodes **102**. As mentioned previously, the smallest unit of protection in most conventional XML access control models has been the node **102**. This method of protection, however, may violate various security principles such as the "need-to-know" and "choice" security principles by leaking unnecessary or confidential information.

[0018] For example, consider the sub-tree rooted at node **102***a* and represented by the literal "VIP Accounts." Suppose that the security policy is such that access to node **102***b* is authorized while access to node **102***c* is unauthorized. Using a node-based security approach, granting access to node **102***b* will normally require granting access to the root node **102***a*. Once access is granted to the root node **102***a*, access will normally be automatically granted to the child node **102***c*. Thus, it may be very difficult to implement a node-based security approach that can grant access to node **102***b* while simultaneously denying access to node **102***c*. As a result, many node-based security approaches violate the "need-to-know" or "choice" security principles because they leak information about the node **102***c*.

[0019] In selected embodiments in accordance with the invention, a path- or relationship-based security approach may be used to provide a more fine-grained, expressive, and effective access control model to protect information in the XML document **100**. In such a model, ancestor/descendant and sibling relationships **104**, or paths **104**, may be considered legitimate elements to be protected. Such a model may also better comply with security principles such as the "need-to-know" and "choice" security principles.

[0020] In certain embodiments, one or more of the paths **104** may be protected with a "security label" associated with a label-based access control (LBAC) implementation. In such an implementation, the path security label may be compared to an access security label granted to a subject (e.g., a user) attempting to access or traverse the path. Whether access is authorized may be determined based on pre-determined set of label access rules. Access to the path may then be denied or granted based on the label access rules.

[0021] For example, consider again the sub-tree rooted at node **102***a*. If the security policy is such that access to node **102***b* is authorized while access to node **102***c* is unauthorized, a security label **106***a* may be attached to the parent-to-child path between the root node **102***a* and the child node **102***c*. A second security label **106***b* may be attached to the sibling-to-sibling relationship between node **102***c* and node **102***b*. As a result, access may be granted to the path between the root node **102***a* and the child node **102***b* while simultaneously denying access to all paths leading to the child node **102***c*.

[0022] In selected embodiments, an SQL extension, also referred to herein as a command or statement, may be provided to enable an access security label to be granted to a user. Such an extension may already be available in various database management systems, such as IBM's DB**2** version **9**. For example, one embodiment of an extension may be implemented using the following SQL statement:

```
GRANT ACCESS LABEL label-name
TO USER user-name FOR READ ACCESS
```

Here, label-name designates the name of the access security label and user-name designates the name of the user who is granted the access security label. Similarly, the phrase "FOR READ ACCESS" may be replaced with the phrase "FOR WRITE ACCESS" or "FOR ALL ACCESS" to grant either read access, write access, or both types of access to the user.

[0023] Referring to FIGS. **2** and **3**, various SQL/XPath extensions may also be provided to enable security labels to be attached to paths **104** between nodes **102**. For example, FIG. **2** shows one embodiment of an SQL statement that may be used to attach a security label to an ancestor/descendant path (including a parent-to-child path) of an XML document. In this embodiment, path**1** and path**2** are XPath expressions designating the nodes at each end of the path, with path**2** being an XPath expression relative to path**1**. Path-label may be used to designate the name of the security label that is attached to the path.

[0024] For example, the following statement may be used to attach a path security label having the name "EXISTENCE" to the relationship between the node **102***a* and the node **102***c* of FIG. **1**:

```
ATTACH EXISTENCE
ANCS //VIP Accounts
DESC /Account[Customer/Name = "Barbara"]
```

[0025] FIG. **3** shows one embodiment of an SQL statement that may be used to attach a security label to a sibling-to-

sibling path of an XML document. In this embodiment, path**1**, path**2**, and path**3** are XPath expressions, with path**2** and path**3** being XPath expressions relative to path**1**. Path**2** and path**3** specify relationships between the node specified by path**1**, and the node's preceding and following siblings. If the node does not have preceding siblings, the PRECEDING-SIBLING expression may be deleted from the statement. Similarly, if the node does not have following siblings, the FOLLOWING-SIBLING expression may be deleted from the statement. Like the extension illustrated in FIG. **2**, path-label may designate the name of the security label attached to the sibling-to-sibling path.

[0026] For example, the following statement may be used to attach a path security label with the name "VALUE" to the sibling-to-sibling relationship between the node **102***b* and the node **102***c* of FIG. **1**:

```
ATTACH VALUE
NODE //Account[Customer/Name = "Barbara"]
PRECEDING SIBLING /Account
```

[0027] In addition to providing support for the above SQL/XPath statements, an extension may be provided to the SQL compiler. This extension may ensure that the access plan generated to fetch a column of type XML in a database table also includes the access rules for evaluating a user's access rights with respect to the content of the XML column. The goal is to allow users to label node relationships and let them be sure that what they want to conceal is truly concealed from the users whose access labels do not satisfy the label access policy with the path labels. Unfortunately, it is impossible to guarantee concealment for any arbitrary set of relationships. Sometimes, it is possible to infer a concealed relationship from the relationships that are not concealed.

[0028] Let us consider an example of four cases where a relationship could be inferred from a pair of non-concealed relationship. Referring to FIG. **1**, suppose it is known that Account Node **102***b* is a descendant of VIP Accounts Node **102***a* and Customer Node **102***d* is a descendant of Account Node **102***b*. Then, there is no point to conceal the ancestor-descendant relationship between VIP Accounts Node **102***a* and Customer Node **102***d*. Suppose it is known that Customer Node **102***d* is a descendant of VIP Accounts Node **102***a* as well as Account Node **102***b*. Since there is only one path from the root of the document to Account Node **102***b*, there is no point to conceal the ancestor-descendant relationship between VIP Accounts Node **102***a* and Account Node **102***b*.

[0029] Suppose it is known that Account Node **102***b* and Account Node **102***c* are the children of VIP Accounts Node **102***a*, then there is no point to conceal the sibling relationship between Account Node **102***b* and Account Node **102***c*. Suppose it is known that VIP Accounts Node **102***a* has a descendant Customer Node **102***d* and the customer has a sibling Account ID **102***e*, then there is no point to conceal the ancestor-descendant relationship between VIP Accounts Node **102***a* and Account ID **102***e*. We say a set of labeled relationships/paths in an XML document D is not secure with respect to a path label L if one of the following four cases occurs.

[0030] 1. Case 1: D has three nodes, n_1, n_2 and n_3 s.t. the ancestor-descendant path from n_1 to n_2 and the ancestor-descendant path from n_2 to n_3 have labels $L_{12}<L$ and $L_{23}<L$. The ancestor-descendant path from n_1 to n_3 has a label $L_{13}\geqq L$.

[0031] 2. Case 2: D has three nodes, n_1, n_2 and n_3 s.t. the ancestor-descendant path from n_1 to n_3 and the ancestor-descendant path from n_2 to n_3 have labels $L_{13}<L$ and $L_{23}<L$. The ancestor-descendant path from n_1 to n_2 has a label $L_{12}\geqq L$.

[0032] 3. Case 3: D has three nodes, n_1, n_2 and n_3 s.t. n_1 is the parent of n_2 and n_3, the parent-child path from n_1 to n_2 and the parent-child path from n_1 to n_3 have labels $L_{12}<L$ and $L_{13}<L$. The sibling path from n_2 to n_3 has a label $L_{23}\geqq L$ or the sibling path from n_3 to n_2 has a label $L_{32}\geqq L$.

[0033] 4. Case 4: D has three nodes, n_1, n_2 and n_3 s.t. the ancestor-descendant path from n_1 to n_2 has a label $L_{12}<L$, and either the sibling path from n_2 to n_3 has a label $L_{23}<L$ or the sibling path from n_3 to n_2 has a label $L_{32}<L$. The ancestor-descendant path from n, to n_3 has a label $L_{13}\geqq L$.

[0034] There is a simple test to verify that a set of labeled relationships/paths in an XML document D is not secure with respect to a path label L. The test starts by computing three ternary relations R_1, R_2 and R_3. The first two columns store the start/end nodes of paths. The third column stores the label associated with paths (if a label is missing, then it is a NULL value). In particular, R_1 stores all ancestor-descendant paths in D, R_2 stores all parent-child paths in D, and R_3 stores all sibling paths in D.

[0035] 1. Case 1 is true for a path label L if and only if the expression $\pi_{\$1,\$5}(R_{1,L}{}^{*}\$2{=}\$1\ R_{1,L}){-}R_{1,L}$ is not empty where $R_{1,L}$ is $\sigma_{\$3<L}(R_1)$.

[0036] 2. Case 2 is true for a path label L if and only if the expression $\pi_{\$1,\$4}(R_{1,L}{}^{*}\$2{=}\$2\ R_{1,L}){-}R_{1,L}$ is not empty where $R_{1,L}$ is $\sigma_{\$3<L}(R_1)$.

[0037] 3. Case 3 is true for a path label L if and only if the expression $\pi_{\$2,\$5}(R_{2,L}{}^{*}\$1{=}\$1\ R_{2,L}){-}R_{3,L}$ is not empty where $R_{2,L}$ is $\sigma_{\$3<L}(R_2)$ and $R_{3,L}$ is $\sigma_{\$3<L}(R_3)$.

[0038] 4. Case 4 is true for a path label L if and only if the expression $\pi_{\$1,\$5}(R_{1,L}{}^{*}\$2{=}\$1\ R_{3,L}){-}R_{1,L}$ is not empty where $R_{1,L}$ is $\sigma_{\$3<L}(R_1)$ and $R_{3,L}$ is $\sigma_{\$3<L}(R_3)$.

[0039] Furthermore, we give intuitive conditions to construct a secure set of labeled relationships for an XML document. If we ignore the directions of ancestor-descendant and sibling paths, all these paths form cycles in an XML document. To assign a path label L to a relationship between two nodes n_1 and n_2 in an XML document D, we must make sure, for every cycle that includes the path from n, to n_2, either there is another path whose label $L_o\geqq L$, or n_1 and n_2 are descendants of some nodes in the cycle and n_1, n_2 are not children of the same parent. Both cases ensure there is uncertainty whether a relationship between two nodes n_1 and n_2 exists: the first case by having another path missing in the cycle, while in the second case, the fact that n_1 and n_2 are descendants of some nodes in the cycle introduces uncertainty except when they are children of the same parent, in which case the sibling relationship between n_1 and n_2 is leaked.

[0040] In certain embodiments, a DTD may be used to verify that certain security labels are assigned to paths of an XML document **100**. In the event one or more paths of an XML document **100** do not include the security labels specified in the DTD, these security labels may be added to the XML document **100** to make it conform to the DTD. This feature may be provided to ensure that protected information in an XML document **100** is truly concealed from users lacking the required authority. This feature may also reduce the chance that users will infer the existence of a concealed relationship from other relationships that are not concealed.

[0041] For example, in certain embodiments, security labels may be validated in an XML document **100** using an attribute declaration in a DTD having the following form:

```
<!ATTLIST N SecurityLabel (Path1 Label1 | Path2 Label2 | . . .),
#REQUIRED/#IMPLIED>
```

[0042] Here, N can be instantiated to be a set of nodes in an XML document **100** (e.g., VIP Accounts), Path**1**, Path**2**, etc. identify instantiated paths relative to each of the nodes to be protected by a security label, and Label**1**, Label**2**, etc. identify security labels to be attached to the instantiated paths of Path**1**, Path**2**, etc., respectively. In selected embodiments, N, Path**1**, Path**2**, etc. may be identified using XPath expressions. Similarly, Path**1**, Path**2**, etc. may designate ancestor/descendant, sibling-to-sibling, or other desired paths in the XML document **100**. The #REQUIRED/#IMPLIED syntax may be used to designate whether the security labels identified in the attribute declaration are required (e.g., #REQUIRED) or are merely optional (e.g., #IMPLIED).

[0043] In operation, when validating an XML document **100** with the DTD, the above-identified attribute declaration may be checked against the attributes in the XML document **100**. This may be performed to verify that the XML document **100** has path security labels at least as restrictive as those designated in the DTD. If the XML document **100** does not include path security labels that are at least as restrictive as those designated in the DTD, path security labels may be inserted into the XML document **100** to make it conform to the DTD. Conversely, path security labels of the XML document **100** that are more restrictive than those designated in the DTD may be left alone. Thus,

the DTD may be used to impose a set of minimum security requirements on paths of the XML document **100**.

[0044] In certain embodiments, when attempting to access an XML document **100**, a user's security label may be compared to the path security labels designated in the DTD as opposed to comparing it with the path security labels of the XML document **100**. This may improve efficiency because a DTD is typically much smaller than the XML document **100** it is associated with. If the user is not authorized to access the paths specified in the DTD, the user will not be authorized to access the corresponding instantiated paths in the XML document **100**. This is because the XML document **100** will have security labels that are at least as restrictive as those specified in the DTD.

[0045] On the other hand, if the user is authorized to access paths designated in the DTD, the user is not necessarily authorized to access the corresponding paths in the XML document **100**. This is because the XML document **100** may have security labels that are more restrictive than those specified in the DTD. If this is the case, the user's security label may also be compared to the path security labels of the XML document **100** to determine whether the user is authorized to access the paths.

[0046] It should be recognized that the features and advantages discussed above with respect to a DTD may also be applied to other languages for describing the schemas of XML documents, such as the XSD language.

[0047] The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed is:

1. A computer program product comprising a computer readable medium having: computer usable program code executable to perform operations to control access to an XML document comprising a plurality of nodes and a plurality of paths between each of the nodes, the operations of the computer program product comprising:

referencing a schema definition comprising a path security label definition associated with a sibling-to-sibling path of an XML document;

receiving an XML document to be validated by comparison with the schema definition;

comparing the XML document to the schema definition;

verifying that the XML document has a path security label associated with a sibling-to-sibling path that is at least as restrictive as that specified by the path security label definition of the schema definition for the nodes associated with the sibling-to-sibling path;

determining an access security label assigned to a user seeking to access the sibling-to-sibling path protected by the path security label;

comparing, using pre-determined access rules, the path security label to the access security label to determine whether the user is authorized to access the sibling-to-sibling path; and

controlling access to the sibling-to-sibling path in accordance with the access rules.

2. The computer program product of claim **1**, wherein the sibling-to-sibling path is specified in the schema definition using at least one XPath expression.

3. The computer program product of claim **1**, wherein the schema definition is selected from the group consisting of a document type definition (DTD) and an XML schema definition (XSD).

4. The computer program product of claim **1**, wherein the access security label assigned to a user is assigned by a user issuing an SQL command utilizing an SQL extension to assign the access security label.

5. A computer program product to control access to an XML document comprising a plurality of nodes and a plurality of paths between each of the nodes, the computer program product comprising a computer-readable medium storing a program of computer-readable instruction that when executed on a computer causes the computer to:

generate a schema definition comprising a path security label definition associated with a sibling-to-sibling path of an XML document;

receive an XML document to be validated by comparison with the schema definition;

compare the XML document to the schema definition;

verify that the XML document has a path security label associated with a sibling-to-sibling path that is at least as restrictive as that specified by the path security label definition;

assign an access security label to a user seeking to access the sibling-to-sibling path protected by the path security label;

compare, using pre-determined access rules, the path security label to the access security label to determine whether the user is authorized to access the sibling-to-sibling path; and

control access to the sibling-to-sibling path in accordance with the access rules.

6. The computer program product of claim **5**, wherein the sibling-to-sibling path is specified in the schema definition using at least one XPath expression.

7. The computer program product of claim **5**, wherein the schema definition is selected from the group consisting of a document type definition (DTD) and an XML schema definition (XSD).

8. The computer program product of claim **5**, wherein assigning an access security label comprises utilizing an SQL extension to assign the access security label.

* * * * *